Министерство образования
Российской Федерации

Московский государственный университет леса

А. В. Чернышов

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Лабораторный практикум
для студентов специальности 2201.00

Издательство Московского государственного университета леса
Москва — 2000

УДК 681.3

6Л2 Чернышов А. В. Системное программное обесречение: Лабораторный практикум для студентов специальности 2201.00. — М.: МГУЛ, 2000. — 64 с.: ил.

В лабораторном практикуме дан необходимый минимум информации для выполнения лабораторных работ по дисциплине «Системное программное обеспечение», а также индивидуальные задания на каждую лабораторную работу и требования к оформлению отчётов по ним.

Одобрено и рекомендовано к изданию редакционно-издательским советом университета

Рецензент — доцент кафедры вычислительной техники Ю. А. Федотов.

Кафедра вычислительной техники

Автор — Александр Викторович Чернышов, доцент

Редактор Е. Г. Петрова

По тематическому плану внутривузовских изданий учебной литературы на 2000 г., доп.

Оригинал-макет выполнен в пакете *CyrTUG*-EmTEX с использованием кириллических шрифтов семейства LH.

Вёрстка в TEXe: А. В. Чернышов

Лицензия 020718 от 02.02.1998 г.

Подписано к печати Тираж 100 экз.

Объем 4 п. л. Зак.

Издательство Московского государственного университета леса.

141005. Мытищи-5, Московская обл., 1-я Институтская, 1, МГУЛ.

Телефон: (095) 588–57–62

Введение

Настоящий лабораторный практикум включает все лабораторные работы, которые должны быть выполнены студентами при изучении дисциплины «Системное программное обеспечение», занимающей два учебных семестра. К первому семестру относятся лабораторные работы с первой по шестую, ко второму — с седьмой по одиннадцатую.

Для выполнения большинства работ студентам необходимо знание языков программирования Си и Ассемблер ПЭВМ IBM PC. Предполагается, что студенты изучили эти языки на предыдущих курсах.

К каждой лабораторной работе даётся необходимый минимум теоретического материала. В некоторых случаях этот минимум ограничивается общими замечаниями и ссылками на общедоступную литературу. Однако рекомендуется познакомиться с литературой и по тем работам, к которым дан развёрнутый материал.

К каждой лабораторной работе для студентов одной группы подобраны по возможности индивидуальные задания.

Все работы, за исключением первой, второй и восьмой, могут быть выполнены под управлением операционных систем UNIX (Linux) или MS DOS. Первая и вторая работы должны выполняться в ОС UNIX. Восьмая лабораторная работа предполагает использование ОС MS DOS и транслятора с языка Ассемблера ПЭВМ IBM PC.

Лабораторная работа 1
Применение фильтров ОС UNIX

1.1. Общие сведения

UNIX — многозадачная многопользовательская операционная система разделения времени. В настоящий момент UNIX-подобные ОС реализованы практически на всех аппаратных платформах от супер-ЭВМ Cray до ПЭВМ типа IBM PC. Практически все они функционируют по единым принципам и предоставляют пользователям «стандартный» интерфейс.

Научившись работать на одной из реализаций, пользователь без особого труда сможет решать свои задачи на любой другой реализации. Эта возможность в частности обеспечивается:

– единым набором программ-фильтров для решения наиболее часто встречающихся задач;

– единым способом изменения поведения программ путём использования ключей;

– единым принципом построения конвейеров из программ-фильтров и переадресации ввода-вывода программ.

Лабораторные работы выполняются студентами на ОС Linux — свободно распространяемой реализации ОС UNIX.

Интерпретатор shell

Важнейшей компонентой стандартного программного обеспечения ОС UNIX, обеспечивающей взаимодействие ОС и пользователя, является интерпретатор shell. Именно этот интерпретатор обеспечивает такие функции ОС UNIX как перенаправление ввода-вывода, построение конвейеров программ и возможность задания сокращённых имён файлов.

В настоящее время существует довольно большое число различных по назначению и возможностям реализаций этого интерпретатора. К наиболее популярным относятся `tcsh`, `ksh`, `bash`. Для подавляющего большинства реализаций UNIX интерпретатором по умолчанию, обеспечивающим выполнение системных командных скриптов, остаётся `sh`.

В ОС Linux наиболее часто используемым является интерпретатор `bash`. Он полностью совместим снизу вверх с интерпретатором `sh`.

Для выполнения лабораторных работ не потребуется знания специфических особенностей `bash`. Необходимы только общие сведения об интерпретаторе shell.

Основное назначение интерпретатора — выполнение команд, введённых пользователем в командной строке. Ввод команд осуществляется после получения на экране приглашения, которое зависит от интерпретатора и окружения пользователя и может быть пользователем изменено. Пользователь может ввести в командную строку: – одну команду UNIX;

– группу независимых команд, разделённых символом `;`, которые будут выполнены последовательно и независимо друг от друга, после чего пользователь снова получит приглашение;

– группу команд, объединённых в конвейер.

В качестве аргументов команд, как правило, выступают имена файлов. Эти имена могут быть заданы как полностью, так и сокращённо с помощью шаблонов. В общем случае применяется три способа сокращения:

– символ `*` задаёт любое количество произвольных символов;

– символ `?` задаёт любой одиночный символ;

– группа символов, записанных в квадратных скобках `[` и `]` задаёт любой символ из перечисленных в скобках.

Шаблоны сопоставляются только с именами существующих файлов. Поэтому команды группового переименования, заданные из командной строки «в лоб», работать не будут.

Специальное значение символов шаблонов можно отменить, предварив каждый такой символ символом `\` (например `\?`) или заключив его в одинарные прямые кавычки (например `'*'`).

Запущенная на выполнение команда называется процессом. Для каждого процесса в ОС UNIX создаётся три стандартных файла (потока):

– стандартный файл ввода `stdin`;

– стандартный файл вывода `stdout`;

– стандартный файл сообщений об ошибках (диагностики) `stderr`.

По умолчанию `stdin` связан с клавиатурой, `stdout` и `stderr` — с экраном терминала пользователя. Это соглашение можно изменить. Символ `>` используется для перенаправления потока `stdout` в файл с указанным именем. При этом, если файл с таким именем существует, он уничтожается. Если необходимо добавить данные в файл, то

применяется комбинация `>>`.

Символ `<` используется для переключения потока `stdin` на файл с указанным именем.

Для перенаправления стандартного потока ошибок служит комбинация `2>`. Если необходимо объединить стандартный поток ошибок со стандартным потоком вывода, применяют комбинацию `2>&1`.

Примеры:

```
ls
```

выдать список файлов текущего подкаталога на терминал;

```
ls >list
```

выдать список файлов текущего подкаталога в файл `list`;

```
ls >>list
```

добавить список файлов текущего подкаталога в конец файла `list`.

Конвейер представляет собой совместный запуск нескольких стандартных программ-фильтров, при котором стандартный вывод `stdout` одной программы поступает на стандартный ввод `stdin` другой. Таким образом удаётся решать многие сложные задачи по обработке информации как последовательность простых задач. Передача данных по конвейеру задаётся символом `|`.

Обычно аргументами команд считаются строки, разделённые пробелами и табуляциями. Но в некоторых случаях необходимо, чтобы пробелы и табуляции были частью аргумента. Это достигается использованием кавычек. В командной строке могут использоваться три типа кавычек:

– двойные кавычки (`"..."`) — задают аргумент, в котором интерпретатором выполняются все раскрытия символов шаблонов (`*`, `?` и т. п.);

– одинарные прямые кавычки (`'...'`) — задают аргумент, передаваемый программе без обработки интерпретатором (в нём сохраняются все специальные символы);

– одинарные обратные кавычки (`` `...` ``) — команда, заключённая в эти кавычки, выполняется, и её результат подставляется в качестве аргументов в командную строку.

Получение подсказки: man

В процессе работы в ОС UNIX пользователь может получить подробную справочную информацию практически по любой стандартной команде. Для этого служит команда `man`. Это одна из немногих команд, предполагающих интерактивный режим работы с пользователем по умолчанию. Формат её вызова:

```
man command
```
где `command` — команда, по которой необходима справка. Информация выдаётся на экран постранично. Для перехода к следующей странице достаточно нажать [ ]*, для выхода из справки — [q]. Можно попытаться найти нужную информацию, не пролистывая всю справку. Для этого необходимо нажать [/] и ввести слово для поиска, после чего нажать [Enter]. Повтор поиска — [n].

К сожалению, в большинстве случаев подсказка даётся на языке оригинала (английском).

Команда ls

Команда предназначена для просмотра содержимого подкаталога. По умолчанию выдаётся содержимое текущего подкаталога. Но можно просмотреть содержимое любого подкаталога, задав его в качестве аргумента. Задав в качестве аргумента шаблон, можно просмотреть только файлы, соответствующие шаблону.

Наиболее употребительные ключи:

`-l` список файлов подкаталога с дополнительной информацией о каждом файле;

`-a` список файлов, включая файлы, имена которых начинаются с точки (`.`);

`-d` список файлов, в котором подкаталоги распечатываются как обычные файлы без распечатки их содержимого;

`-t` список файлов, упорядоченный по времени создания (начиная с новых);

`-u` список файлов, упорядоченный по времени последнего использования;

`-r` упорядочивание файлов производится в обратном порядке.

Ключи можно комбинировать для получения соответствующего эффекта.

Примеры:

```
ls
```
просмотр содержимого текущего подкаталога;
```
ls -l
```
просмотр содержимого текущего подкаталога с дополнительной информацией о файлах, в частности, с информацией о размерах файлов, их владельцах, правах доступа;
```
ls -a
```

---

* Клавиша «Пробел».

просмотр содержимого текущего подкаталога, включая все файлы, имена которых начинаются с точки;

```
ls -la
```

комбинация двух предыдущих команд;

```
ls /usr
```

просмотр содержимого подкаталога `/usr`;

```
ls m*
```

вывод имён файлов текущего подкаталога, начинающихся с буквы `m`, (таких файлов может и не быть).

Команда cat

Команда предназначена для объединения содержимого файлов, заданных в качестве аргументов и вывода результата в стандартный поток вывода.

Пример:

```
cat tail.1 tail.2 tail.3 >all.4
```

команда объединяет содержимое файлов `tail.1`, `tail.2` и `tail.3` и помещает результат в файл `all.4`.

Команда echo

Команда дублирует свои аргументы в стандартный поток вывода. Очень удобна для проверки того, во что раскрываются шаблоны в командной строке. Например:

```
echo *
```

просто выдаст список файлов в текущем подкаталоге.

Команда cp

Команда предназначена для копирования файла в файл с другим именем или для копирования группы файлов в указанный подкаталог.

Полезный ключ:

`-r` копировать с деревом подкаталогов.

Примеры:

```
cp файл_источник файл_приёмник
```

копирует `файл_источник` в `файл_приёмник`;

```
cp список_файлов подкаталог
```

копирует `список_файлов` в `подкаталог`.

Команда mv

Команда предназначена для переименования файла или для перемещения группы файлов в указанный подкаталог.

Примеры:

`mv файл_источник файл_приёмник`

переименовывает `файл_источник` в `файл_приёмник`;

`mv список_файлов подкаталог`

перемещает `список_файлов` в `подкаталог`.

Команда rm

Команда предназначена для стирания файла или группы файлов. Для удаления подкаталога используется команда `rmdir`. При этом удаляемый подкаталог должен быть пустым.

Команда ps

Команда предназначена для получения информации о запущенных в системе процессах. Может быть полезна для наблюдения за работой фоновых процессов.

Полезные ключи:

`-a` показывать процессы всех пользователей;

`-x` показывать процессы, не привязанные к терминалам.

Классической для системных администраторов является команда:

`ps -ax`

показывающая список всех процессов в системе, включая процессы, не привязанные к терминалам.

Фильтр wc

Фильтр предназначен для подсчёта числа строк, слов и символов в стандартном потоке ввода. По умолчанию выдаёт все три значения в стандартный поток вывода. Если в качестве аргумента задано имя файла, то вместо стандартного потока ввода используется указанный файл.

Ключи:

`-l` выдать только число строк;

`-w` выдать только число слов;

`-c` выдать только число символов.

Фильтр tail

Фильтр предназначен для вывода в стандартный поток вывода нескольких последних строк стандартного потока ввода. Если в качестве аргумента задано имя файла, то вместо стандартного потока ввода используется указанный файл. По умолчанию выдаёт 10 строк.

Полезный ключ:

-n задаёт число выводимых строк (например `-n 5` — пять строк);

Фильтр head

Фильтр полностью аналогичен фильтру `tail`, за исключением того, что выдаёт в стандартный поток вывода не последние, а первые строки стандартного потока ввода.

Фильтр more

Фильтр предназначен для постраничной выдачи на экран стандартного потока ввода. Работает в интерактивном режиме. После выдачи первой страницы ждёт команды пользователя. Для перехода к следующей странице достаточно нажать [ ], для прекращения просмотра и выхода в shell — [q]. Можно попытаться найти нужную информацию, не пролистывая все страницы. Для этого необходимо нажать [/] и ввести слово для поиска, после чего нажать [Enter]. Повтор поиска — [n]. По достижении конца потока автоматически завершает работу и выходит в shell.

Фильтр grep

Фильтр предназначен для поиска в стандартном потоке ввода или в списке файлов строк, соответствующих регулярному выражению, заданному первым аргументом командной строки:

```
... | grep выражение
```

или

```
grep выражение список_файлов
```

По умолчанию печатает все найденные строки с указанием имён файлов.

Подробное описание правил задания регулярных выражений см. в [4] с. 32 – 34 или в интерактивном справочнике UNIX:

```
man grep
```

Наиболее используемые ключи:

`-i` игнорировать регистр букв;

`-n` дополнительно печатать номера строк в каждом файле;

`-l` печатать только имена файлов, в которых обнаружены строки, соответствующие регулярному выражению;
`-L` печатать только имена файлов, в которых ничего не обнаружено;
`-h` печатать найденные строки без имён файлов.

Фильтр sort

Фильтр предназначен для сортировки строк стандартного потока ввода и выдачи результата в стандартный поток вывода. По умолчанию сортирует строки в алфавитном порядке по возрастанию с первой позиции строки с учётом регистра. Если в качестве аргумента задано имя файла, то вместо стандартного потока ввода сортирует указанный файл.

Полезные ключи:
`-n` сортировать как числа;
`-r` сортировать в обратном порядке;
`-f` сортировать без учёта регистра;
`+P` сортировать, начиная с колонки с номером `P` и до конца строки; колонки нумеруются с 0;
`-P` сортировать, учитывая колонки до (но не включая) `P`.

Фильтр awk

Фильтр предназначен для преобразования информации в файлах (стандартном потоке ввода), имеющих табличную структуру. Фильтр имеет мощный язык, на котором задаётся программа преобразования. Обработка выполняется построчно. В результате обработки генерируется отчёт в заданной программистом форме. Программы на `awk` можно эффективно использовать в конвейерах для преобразования стандартного потока вывода одной программы и передачи результатов фильтрации на стандартный поток ввода другой программы.

Довольно подробное описание `awk` приведено в [2]. Наиболее полное описание как обычно можно получить (на английском языке) по команде

```
man awk
```

Здесь дадим только краткое описание основных его элементов.

Запуск `awk`:

```
awk программа файлы
```

или

```
... | awk программа
```

Если `программа` очень большая, то её можно записать в отдельный файл и вызвать следующим образом:

```
... | awk -f файл_с_программой
```

Каждая строка файла считается записью, состоящей из полей. Поля по умолчанию разделяются символами пробелов и табуляций. Они автоматически нумеруются, начиная с единицы. Внутри программы поля доступны как `$1`, `$2` и т. д. Количество распознанных в текущей строке полей хранится в предопределённой переменной `NF`. Строка целиком доступна как `$0`.

Программа на `awk` записывается как последовательность блоков, каждый из которых имеет вид:

```
селектор{
действия
}
```

где `селектор` используется для выделения строк, к которым будут применены `действия`. Селектор может содержать регулярные выражения, проверки условий и их комбинацию. Он может отсутствовать. Тогда действия блока выполняются над всеми строками файла.

Действия записываются как последовательность операторов, каждый из которых обязательно заканчивается символом `;`.

Существует два селектора специального вида:

`BEGIN` — задаёт блок действий, которые будут выполнены перед чтением первой строки;

`END` — задаёт блок действий, которые будут выполнены после чтения последней строки.

Оба селектора являются необязательными.

При разработке программы программист может использовать собственные переменные и массивы. Все переменные и массивы не требуют предварительного определения и начинают существовать в момент первого использования. По умолчанию все они имеют строковый тип — каждая переменная и каждый элемент массива могут хранить строку символов произвольной длины. Но если строки представляют собой числа (последовательность цифр), то над ними можно выполнять арифметические действия. При этом результат действия снова может быть обработан как строка.

Значением любой пользовательской переменной по умолчанию является в зависимости от контекста пустая строка (`""`) или число ноль (`0`).

Над числами допускаются любые арифметические операции, допустимые в языке Си. Можно также выполнять их сравнение.

Сравнение на равенство и неравенство допустимо также для строк. Кроме этого существует специальная операция соответствия строки заданному шаблону, которая записывается как `~` (и операция несоответствия — `!~`).

Массивы могут быть ассоциативными, то есть в качестве индексов у них могут выступать не только числа, но и строки символов. Для выполнения цикла по ассоциативному массиву существует специальный оператор:

```
for(i in ms){
действия
}
```

где `i` — индексная переменная цикла, а `ms` — ассоциативный массив.

Существуют также операторы `for`, `if else`, `while`, соответствующие по синтаксису и назначению аналогичным операторам Си.

Вывод результатов работы осуществляется оператором `printf`, записываемым в соответствии с правилами языка Си.

В языке `awk` предусмотрено несколько полезных функций:

`length(строка)` — возвращает длину строки;

`substr(S,M,N)` — возвращает часть строки `S`, начинающуюся от позиции `M` и имеющую длину не более `N` символов. Символы в строке нумеруются с `1`. Если аргумент `N` не указан, то возвращаются все символы до конца строки;

`split(S,M,N)` — разбить строку `S` на подстроки, которые поместить в массив `M`. Границы разбиения задаются регулярным выражением `N`. Функция возвращает количество элементов в массиве `M`;

`index(S,N)` — возвращает номер позиции, с которой строка `N`, совпадает со строкой `S`. Если совпадения нет, то возвращается `0`;

`tolower(S)` — возвращает строку `S`, все символы которой приведены к нижнему регистру;

`toupper(S)` — возвращает строку `S`, все символы которой приведены к верхнему регистру;

`system(command)` — выполняет команду системы `command` в порождённой оболочке, после чего возвращает управление в программу `awk`.

Имеется ряд других функций, в том числе для выполнения некоторых математических операций, преобразования строк и времени, форматирования вывода.

В заключение описания покажем, как на `awk` можно реализовать программу подсчёта числа строк, слов и символов в файле:

```
awk '{
    cntb+=length($0);
    cntw+=NF;
    cntl++;
}
END{
    printf "Строк: %d, слов: %d, символов: %d\n",
                          cntl,cntw,cntb;
}'
```

Обрабатываемый файл должен быть либо передан по конвейеру, либо указан в командной строке.

Универсальный подход к решению сложных задач в ОС UNIX

Основная идея решения сложных задач в ОС UNIX заключается в разбиении такой задачи на последовательность простых шагов, которые могут быть выполнены с использованием уже имеющихся в системе стандартных фильтров. В крайнем случае разрабатываются несложные фильтры к тем шагам, для которых нет подходящих фильтров. Далее фильтры просто объединяются в конвейер — и задача решена.

Приведём пример. Дан текстовый файл `text.txt`, содержащий некоторый текст. Необходимо вычислить длину его 36-й строки. Один из вариантов решения заключается в выделении 36-й строки и использовании фильтра `wc` для подсчёта числа символов в строке. Для выделения нужной строки можно воспользоваться комбинацией фильтров `head` и `tail`.

Решение:

```
cat text.txt | head -n 36 | tail -n 1 | wc -c
```

прочитать файл, выделить первые 36 строк и из них выделить последнюю строку, подсчитать число символов в ней. Результат работы конвейера — искомая длина 36-й строки.

Ещё один пример. Необходимо подсчитать, в скольких строках заданного файла содержится слово «UNIX». Для поиска строк, содержащих указанное слово, воспользуемся фильтром `grep`.

Решение:

```
cat text.txt | grep UNIX | wc -l
```

прочитать файл, выделить из него все строки, содержащие слово «UNIX», и подсчитать число выделенных строк. Результат работы конвейера — число строк, содержащих слово «UNIX».

1.2. Задания на лабораторную работу

Каждый студент должен выполнить все общие задания и одно индивидуальное, соответствующее его номеру в журнале.

Общие задания

1. Подсчитать число файлов в подкаталоге.
2. Найти в подкаталоге файл наибольшего объёма.
3. Вычислить суммарный объём, занимаемый файлами в подкаталоге.
4. Вычислить средний объём файла в подкаталоге.

Индивидуальные задания

1. Привести все имена файлов в подкаталоге к нижнему регистру.
2. Привести все имена файлов в подкаталоге к верхнему регистру.
3. Заменить расширение у группы файлов (переименовать файлы).
4. Найти запущенный процесс `httpd`.
5. Вывести список запущенных процессов в алфавитном порядке.
6. Подсчитать число запущенных терминальных процессов (процессов, содержащих в названии `tty`).
7. Подсчитать в подкаталоге число файлов, начинающихся с прописной буквы.
8. Подсчитать в подкаталоге число файлов, начинающихся со строчной буквы.
9. Подсчитать в подкаталоге число файлов, начинающихся с точки.
10. Подсчитать в подкаталоге число файлов, не содержащих в имени точек.
11. Подсчитать в подкаталоге число файлов, содержащих в имени хотя бы одну точку.
12. Подсчитать в подкаталоге число файлов, не являющихся подкаталогами.
13. Подсчитать в подкаталоге число файлов, являющихся подкаталогами.
14. Найти в подкаталоге файл, содержащий наибольшее количество строк.
15. Найти в подкаталоге файл, содержащий наименьшее количество строк.
16. Найти в подкаталоге файл, содержащий наибольшее количество слов.

17. Найти в подкаталоге файл, содержащий наименьшее количество слов.

18. Подсчитать суммарное количество строк во всех файлах подкаталога.

19. Подсчитать суммарное количество слов во всех файлах подкаталога.

20. Подсчитать в файле число слов, начинающихся с прописной буквы.

21. Подсчитать в файле число слов, начинающихся со строчной буквы.

22. Подсчитать в файле число слов, состоящих только из цифр.

23. Найти слово, встречающееся в файле наиболее часто.

24. Подсчитать число запятых в файле.

25. Подсчитать число точек в файле.

1.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание используемых команд (фильтров) и их ключей;

– конвейеры, являющиеся решением задания с комментариями;

– результаты работы конвейеров.

1.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о работе с ОС UNIX на уровне пользователя;

– знать назначение и основные ключи описанных в практикуме команд и фильтров UNIX;

– владеть методами разбиения сложных задач преобразования информации на ряд простых шагов, решаемых существующими фильтрами;

– уметь строить простые конвейеры для решения повседневных задач.

Студент должен быть готов продемонстрировать полученные навыки на защите лабораторной работы по заданию преподавателя.

Лабораторная работа 2
Применение фильтра awk для статистической обработки информации

2.1. Общие сведения

Одной из важнейших задач эксплуатации сервера сети является оценка качества его функционирования. Эта оценка может быть, в частности, получена путём анализа информации об использовании сервера пользователями. Информация такого рода сохраняется службами сервера в специальных файлах, называемых файлами log-ов. Как правило, в файл log-а службой заносится информация о времени обращения пользователя и об оказанных ему услугах.

Собранная в файле log-а информация имеет большой объём и редко может быть использована непосредственно для решения поставленной задачи. Как правило, требуется её статистическая обработка. Для проведения статистической обработки очень удобным средством оказывается фильтр `awk`.

Такая обработка способна дать большой объём качественной информации о работе сервера, о наиболее популярных его ресурсах, об аудитории, посещающей сервер, и о многом другом. Трудно переоценить важность такой информации для думающего администратора.

Рассмотрим файлы log-ов трёх наиболее используемых служб — http, ftp и proxy. Файлы log-ов имеют табличную структуру. Каждая строка файла соответствует записи об одном событии — одном обращении к службе — и может рассматриваться как строка таблицы. Поля в строке разделяются пробелами и могут рассматриваться как колонки таблицы. Это соответствует «представлениям» о таблице фильтра `awk` по умолчанию.

В дальнейшем описании приводится смысл только тех полей, которые будут использоваться при статистической обработке. Номера полей соответствуют их номерам при обработке в `awk`. Конкретный формат каждого поля здесь не приводится, поскольку его легко посмотреть непосредственно в файле log-а.

Структура записи файла log-а proxy

`$1` — время запроса (в секундах в формате системы);
`$5` — объём переданных данных в байтах;
`$6` — тип запроса (чаще всего `GET`);
`$7` — запрошенный ресурс (URL);

`$8` — имя (login) пользователя.

Структура записи файла log-a http

`$1` — запросивший IP адрес;
`$4` — дата и время запроса;
`$6` — метод обращения (чаще всего `GET`);
`$7` — запрошенный ресурс;
`$8` — тип протокола запроса;
`$9` — код завершения операции;
`$10` — объём переданных данных.

Структура записи файла log-a ftp

Здесь рассматривается только структура записи о чтении файла с anonymous ftp.

`$1` — день недели;
`$2` — месяц;
`$3` — число;
`$4` — время;
`$5` — год;
`$7` — запрашивавший IP адрес или доменное имя;
`$8` — полученный объём данных;
`$9` — имя файла с полным путём;
`$14` — пароль, с которым пользователь вошёл на anontmous ftp (обычно его E-mail).

Пример обработки

В качестве примера рассмотрим статистическую обработку информации по файлу log-a proxy-сервера. Файл называется access.log. Считая, что поле `$2` содержит информацию о времени, затраченном proxy на обработку запроса в миллисекундах, найдём суммарное время, затраченное на обработку всех запросов:

```
cat access.log |
awk '{sum+=$2;}END{printf "Всего %f с\n",sum/1000.;}'
```

Как видно из программы, мы читаем файл log-a и построчно обрабатываем его фильтром `awk`. Обработка каждой строки сводится к добавлению значения её второго поля к общей сумме. Когда все строки исчерпаны — файл закончился, — итоговая сумма печатается на стандартный вывод. Для удобства восприятия она приводится к секундам.

2.2. Задания на лабораторную работу

Каждый студент должен выполнить задание, соответствующее его номеру в журнале.

По файлу доступа к proxy-серверу access.log

1. Найти общее (суммарное) число пользователей, пользовавшихся сервером.

2. Найти общее число файлов, запрошенных пользователями.

3. Найти общее число серверов Internet, запрашивавшихся пользователями.

4. Определить суммарный объём информации (в байтах), прошедший через сервер.

5. Определить 10 пользователей, запросивших наибольший объём информации.

6. Определить 10 наиболее активных пользователей (сделавших наибольшее число запросов).

7. Определить 10 серверов Internet, с которых был получен наибольший объём информации.

8. Определить 10 ресурсов Internet (полные URL), пользовавшихся наибольшей популярностью у пользователей — получивших наибольшее количество запросов.

По файлу доступа к http-серверу access_log

9. Определить общее число клиентов, обращавшихся к серверу.

10. Определить общее число файлов, запрошенных с сервера.

11. Определить методы запросов, использовавшиеся при обращении к серверу.

12. Определить протоколы запросов, использовавшиеся при обращении к серверу.

13. Определить общее число успешных запросов к серверу (коды завершения < 400).

14. Определить общее число ошибочных запросов к серверу (коды завершения $\geqslant 400$).

15. Определить 10 наиболее популярных ресурсов на сервере.

16. Определить 10 наиболее активных пользователей сервера.

17. Определить 10 наиболее частых ошибочных запросов к серверу.

По файлу доступа к ftp-серверу xferlog

18. Определить общее число клиентов, обращавшихся к серверу.

19. Определить общее число файлов, запрошенных с сервера.

20. Определить суммарный объём информации, считанный с сервера.

21. Определить объём информации, считанный с сервера каждым пользователем.

22. Определить 10 наиболее популярных ресурсов на сервере.

23. Определить 10 наиболее активных пользователей сервера.

24. Определить пользователей, которые указывают разные пароли при входе.

25. Составить список паролей пользователей.

2.3. Содержание отчёта

Отчёт должен содержать:

- титульный лист;
- задание на работу;
- краткое описание структуры обрабатываемого файла log-а;
- краткое описание используемых операторов `awk`;
- программу на `awk` и конвейер, являющиеся решением задачи;
- результаты работы программы.

2.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о способах и назначении статистической обработки информации файлов log-ов сервера Internet;

– знать основные операторы `awk` и приёмы разработки программ для него;

– владеть методами построения конвейеров для решения задач анализа табличной информации и генерации отчётов.

– уметь определять цели анализа информации файлов log-ов и делать выводы на основе анализа.

Студент должен быть готов продемонстрировать полученные навыки на защите лабораторной работы по заданию преподавателя.

Лабораторная работа 3
Разбор строк ассемблерной программы

3.1. Общие сведения

В данной лабораторной работе мы приступаем к разработке программы простого абсолютного ассемблера. Всего на разработку программы ассемблера отводятся три лабораторные работы — 3, 4 и 5. При этом работы 3 и 4 полностью автономны, а работа 5 включает в себя результаты обеих предыдущих работ. Все три работы выполняются на языке программирования Си.

Теоретический материал будет даваться в разделах «Общие сведения» к тем работам, где он необходим.

Для простоты мы будем разрабатывать ассемблер одноадресной машины. В этом случае в каждой строке ассемблера может содержаться не более одного операнда, что значительно упрощает разбор*.

Типичная программа на ассемблере представляет собой последовательность строк фиксированного формата. Каждая строка состоит в общем случае не более чем из четырёх полей и является либо записью машинной команды, либо директивой ассемблера.

При разборе на поля каждая строка может рассматриваться независимо от других строк. Как правило, в строке ожидаются следующие поля:

`метка операция операнд комментарий`

где под термином `операция` понимается мнемоника машинной операции или директива ассемблера. Под термином `операнд` в случае одноадресной машины будем понимать только имя переменной (ячейки памяти) или метки. Поля разделяются произвольным числом пробелов и табуляций.

Основная сложность разбора такой строки заключается в том, что любое из полей может отсутствовать. Так, например, всю строку может занимать один комментарий, или в строке может быть задана единственная директива ассемблера без параметров. Поэтому для уверенной идентификации полей применяют специальные соглашения, которые, впрочем, разнятся от ассемблера к ассемблеру.

---

* Несколько операндов в одной строке всё же могут содержаться в директиве формирования констант (выделения ячеек памяти с присвоением им значений). Но такой групповой вариант директивы можно просто не рассматривать.

Например, для выделения меток часто применяют символ двоеточия (:), который ставят в конце метки. Но возможен и другой подход, при котором метка обязательно записывается с первой позиции строки, а если в строке метка отсутствует, то такая строка содержит в первой позиции символ пробела или табуляции.

Начало комментария часто отмечается специальным символом (; или /), но может быть введено соглашение, при котором комментарием считается всё, что располагается за операндом или за определённой позицией в строке.

Как правило, в программах на ассемблере не различаются буквы верхнего и нижнего регистров, то есть строки `START`, `Start` и `start` считаются идентичными. Но можно разработать транслятор ассемблера (в тренировочных целях), в котором делается такое различие.

Конечной целью разбора каждой строки является определение типов и количества содержащихся в строке полей и их выделение для последующей обработки.

Можно предложить следующую последовательность разбора строки. Сначала опознаётся и выделяется поле комментария. После этого проверяется наличие метки и производится её выделение. Далее в оставшейся части строки производится поиск и выделение операции (директивы), а затем операнда.

3.2. Задание на лабораторную работу

Разработать программу разбора исходных строк ассемблерной программы на поля. Студент выбирает форму записи исходных строк из табл. 3.1 в соответствии со своим номером в журнале.

Текст тестовой программы студент должен составить самостоятельно. Тестовая программа должна включать в себя несколько машинных команд и основные директивы ассемблера, включая директивы определения констант и резервирования памяти.

Результат разбора каждой строки должен быть напечатан в форме:

`Метка:...  Оператор:...  Операнд:... Комментарий:...`

где под `...` понимается содержание соответствующего поля разбираемой строки.

Таблица 3.1

Номер по журналу	Обозначение метки	Обозначение комментария	Различие регистра	Набор директив по табл. 3.2
1	:	;	Есть	I
2	–	/	Есть	I
3	:	*	Нет	I
4	–	;	Нет	I
5	:	/	Есть	II
6	–	*	Есть	II
7	:	;	Нет	II
8	–	/	Нет	II
9	:	*	Есть	I
10	–	;	Есть	I
11	:	/	Нет	I
12	–	*	Нет	I
13	:	;	Есть	II
14	–	/	Есть	II
15	:	*	Нет	II
16	–	;	Нет	II
17	:	/	Есть	I
18	–	*	Есть	I
19	:	;	Нет	I
20	–	/	Нет	I
21	:	*	Есть	II
22	–	;	Есть	II
23	:	/	Нет	II
24	–	*	Нет	II
25	:	;	Есть	I

Обозначения:

– метка распознаётся по расположению с первой позиции строки;

* комментарий распознаётся по его расположению после заданной позиции в строке.

3.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу, включая описание конкретной формы записи ассемблерной программы;

– схему алгоритма программы разбора входных строк;

Таблица 3.2

Описание директивы	Варианты	
	I	II
Задаёт имя и начальный адрес программы	START	BEGIN
Указывает на конец исходной программы, задаёт первую исполняемую команду программы	END	END
Формирует байтовую (символьную) константу, занимающую столько байтов, сколько необходимо	BYTE	DB
Формирует константу, занимающую одно слово	WORD	DW
Резервирует заданное количество байтов для данных	RESB	RB
Резервирует заданное количество слов для данных	RESW	RW

– описание программы разбора с обоснованием применяемого алгоритма;

– текст программы;

– исходный текст тестовой программы ассемблера и результат работы программы разбора.

3.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о принципах и об основных формах записи ассемблерных программ;

– знать порядок работы с файлами на языке Си;

– владеть методами самостоятельного выбора конкретной формы записи при проектировании ассемблера;

– уметь выделять целевые подстроки из входной строки с применением языка Си.

Лабораторная работа 4
Построение и использование хеш-таблицы

4.1. Общие сведения

Хеширование или **рассеянная память** представляет собой класс методов поиска в таблицах, при которых вместо перебора по некоторому алгоритму элементов таблицы и их сравнения с ключом поиска производится некоторая «свёртка» $h(K)$ ключа K, в результате которой получается индекс, соответствующий положению искомого элемента таблицы.

Во многих случаях хеширование является предпочтительным методом организации поиска информации по таблице, поскольку в среднем обеспечивает поиск нужного элемента за рекордно малое число сравнений с ключом. Однако при сильном заполнении таблицы время поиска конкретного элемента может быть чрезвычайно велико. Поэтому *этот метод не может быть применён в системах реального времени.*

При поиске методом хеширования довольно сложно подобрать такую хеширующую функцию $h(K)$, чтобы для любого K эта функция принимала уникальное значение. Более того, такой подбор в принципе возможен только для заранее известного набора данных в таблице. Поэтому в большинстве случаев приходится мириться с коллизиями, когда $h(K_i) = h(K_j)$ при $K_i \neq K_j$.

При возникновении коллизии необходимо как-то указать способ продолжения поиска по таблице. Методы, позволяющие продолжить поиск в случае коллизии, получили название методов разрешения коллизий.

Следовательно, для использования поиска по хеш-таблице необходимо принять два решения:
- выбрать хеш-функцию $h(K)$;
- выбрать метод разрешения коллизий.

В дальнейшем будем полагать, что хеш-функция имеет не больше M различных значений, которые при всех значениях K удовлетворяют условию $0 \leqslant h(K) < M$. При этом можно понимать M как размер хеш-таблицы.

Выбор хеширующей функции

К хеширующей функции выдвигается два требования:
- её вычисление должно быть очень быстрым;

– она должна минимизировать число коллизий.

Метод деления. Это наиболее известный и простой для понимания метод вычисления хеширующей функции. В этом методе индекс вычисляется как остаток от деления ключа на M:

$$h(K) = K \bmod M.$$

Важным моментом при составлении программы является выбор значения M. При известном наборе входных ключей K удовлетворительное значение M может быть получено экспериментальным путём. При неизвестном наборе входных ключей выбор значения M представляет нетривиальную математическую задачу. Основное требование к значению M при этом — сведение к минимуму числа коллизий.

Доказано, что нельзя выбирать M, равное степени основания системы счисления ЭВМ, а также для буквенных ключей нежелательно значение M, кратное 3. Вообще, лучше всего, если M будет простым числом.

Более подробно вопрос выбора значения M рассмотрен в [5], раздел 6.4.

На языке Си хеш-функция может быть записана как

```
h=K%M;
```

Метод умножения. Этот метод также достаточно прост, но его сложнее понять, чем метод деления. При его рассмотрении необходимо представлять себе, что мы работаем с дробями, а не с целыми числами.

Пусть w — размер машинного слова. Целое число A можно рассматривать как дробь A/w, если мысленно поставить десятичную (двоичную) точку слева от машинного слова, в которое записано A.

Метод состоит в том, чтобы выбрать A взаимно простым с w и положить

$$h(K) = \left\lfloor M \left(\left(\frac{A}{w} K \right) \bmod 1 \right) \right\rfloor .$$

Здесь $\lfloor x \rfloor$ означает «пол от x», наибольшее целое число, не превосходящее x: $\max\limits_{k \leqslant x} k$.

В этом методе значение M выбирают равным степени основания системы счисления ЭВМ. Для привычных нам двоичных ЭВМ это будут степени числа 2. Значение A необходимо выбирать взаимно простым с w. Более подробно см. [5], раздел 6.4.

На языке Си хеш-функция может быть записана как

```
h=(A*K)>>(w-m);
```

Здесь `m` можно найти из выражения $M = 2^m$.

Метод умножения может быть более выгоден на ЭВМ, у которых операция умножения выполняется быстрее операции деления. На многих современных ЭВМ обе операции выполняются за одинаковое время, поэтому обе хеш-функции требуют практически одинаковых затрат машинного времени.

До настоящего момента предполагалось, что ключ K представлен одним машинным словом. Если ключ состоит из нескольких слов, то перед вычислением хеш-функции должно быть выполнено комбинирование слов ключа в одно слово. Комбинирование может быть выполнено сложением слов ключа по модулю w или через «исключающее или». Чтобы избежать одинаковых результатов для комбинаций типа (X, Y) и (Y, X), между сложениями предлагается применять циклический сдвиг.

При отладке программы удобно пользоваться хеш-функцией, у которой $h(K) = 0$ для любого K, чтобы все ключи хранились вместе. Рабочую хеш-функцию можно подставить в программу после отладки.

Выбор метода разрешения коллизий

Здесь будут рассмотрены только простые методы разрешения коллизий. Более подробно с методами разрешения коллизий можно познакомиться в [5], раздел 6.4.

Метод цепочек. При этом методе ключи хранятся в памяти ЭВМ в виде M связанных списков. Как правило, имеется массив размерностью M, содержащий адреса, указывающие на начала списков (ссылки), либо признаки пустых списков. Каждый элемент списка (рис. 4.1) содержит искомый ключ и адрес, указывающий на следующий элемент в списке (либо признак конца списка). Элемент списка может содержать и другую сопутствующую информацию.

Поиск нужного ключа проводится по следующему алгоритму:

1) вычисление хеш-функции и получение индекса выборки из хеш-таблицы;

2) выборка из хеш-таблицы адреса начала цепочки;

3) сравнение ключа поиска с ключом в адресуемом элементе списка;

4) если обнаружено совпадение ключей, то элемент найден и поиск закончен;

5) если ключи не совпадают и в элементе цепочки присутствует

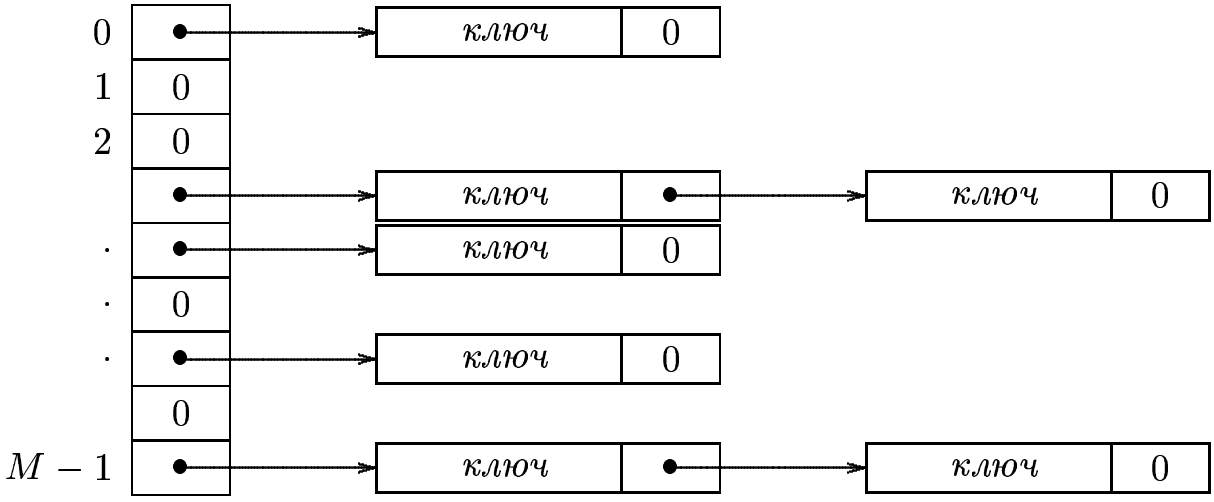

Рис. 4.1. Хранение ключей методом цепочек: 0 — признак конца цепочки; *ключ* — поле для хранения ключа поиска

указатель на следующий элемент, то выбрать адрес элемента и перейти к (3);

6) если в элементе цепочки поле указателя содержит признак конца цепочки, то элемента в таблице нет, и поиск закончен.

Элементы в списках могут быть дополнительно упорядочены по ключам. Однако на практике при этом значительно возрастают затраты на размещение элементов в списке. Время же поиска нужного элемента в среднем растёт очень незначительно.

Рассмотренный метод является не самым оптимальным с точки зрения занимаемой оперативной памяти, поскольку в таблице адресов начал списков может оказаться много незанятых ячеек. В некоторых случаях объём памяти может быть сокращён путём **наложения пространства записей на пространство ссылок**.

В этом случае массив размерностью M содержит сами элементы цепочек. Адреса указателей на следующие элементы ссылаются на сам массив. Это иллюстрирует рис. 4.2.

Как видно из рис. 4.2, при таком способе построения таблицы необходим специальный признак незанятости ячейки таблицы. Это может быть специальное значение поля ключа или специальное значение указателя на следующий элемент цепочки.

Метод «открытой адресации». Этот метод предполагает полный отказ от ссылок на следующие элементы списков. В таблице хранятся только ключи. При совпадении хеш-функций поиск нужного ключа проводится последовательным перебором по определённому алгоритму ячеек таблицы. Если при этом встречается незанятая ячейка, то поиск считается неудачным — искомого ключа в таблице нет.

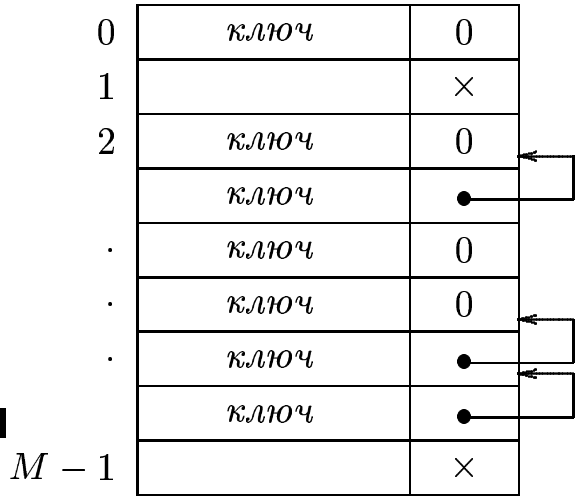

Рис. 4.2. Хранение ключей методом наложения пространства записей на пространство ссылок: 0 — признак конца цепочки; *ключ* — поле для хранения ключа поиска; × — незанятые записи

В простейшем случае применяется линейное опробирование, при котором последовательно проверяются ключи в соседних ячейках таблицы. Если при этом поиск доходит до границы таблицы, то он должен быть продолжен с другого её конца.

При использовании метода «открытой адресации» наблюдается эффект скучивания данных в таблице, при котором одни зоны таблицы заполняются значительно плотнее, чем другие, что замедляет поиск. Несмотря на этот эффект в среднем скорость поиска остаётся довольно высокой, если только заполненность таблицы не превышает 75 %.

Избежать скучивания можно, применяя метод двойного хеширования, который в силу сложности здесь не рассматривается. Этот метод подробно описан в [5], раздел 4.6.

4.2. Задания на лабораторную работу

Целью лабораторной работы является построение хеш-таблицы по выбранному списку ключей и осуществление поиска по таблице выбранного ключа. В качестве ключей рассматриваются мнемокоды команд процессора и директивы ассемблера. Конкретный набор мнемокодов студент выбирает самостоятельно. Список директив берётся из лабораторной работы 3. Хеширующую функцию, метод комбинирования многословных ключей, метод разрешения коллизий и общее количество ключей студент выбирает из табл. 4.1 в соответствии со своим номером в журнале. Необходимые для хеширующей функции константы студент должен подобрать самостоятельно.

Таблица 4.1

Номер по журналу	Хеширующая функция	Комбинирование многобайтовых ключей	Метод разрешения коллизий	Количество ключей
1	%	$\oplus$	$\rightarrow$	17
2	*	$\oplus$	$\hookleftarrow$	17
3	%	$\otimes$	$\uparrow$	17
4	*	$\otimes$	$\rightarrow$	17
5	%	$\oplus$	$\hookleftarrow$	29
6	*	$\oplus$	$\uparrow$	29
7	%	$\otimes$	$\rightarrow$	29
8	*	$\otimes$	$\hookleftarrow$	29
9	%	$\oplus$	$\uparrow$	17
10	*	$\oplus$	$\rightarrow$	17
11	%	$\otimes$	$\hookleftarrow$	17
12	*	$\otimes$	$\uparrow$	17
13	%	$\oplus$	$\rightarrow$	29
14	*	$\oplus$	$\hookleftarrow$	29
15	%	$\otimes$	$\uparrow$	29
16	*	$\otimes$	$\rightarrow$	29
17	%	$\oplus$	$\hookleftarrow$	17
18	*	$\oplus$	$\uparrow$	17
19	%	$\otimes$	$\rightarrow$	17
20	*	$\otimes$	$\hookleftarrow$	17
21	%	$\oplus$	$\uparrow$	29
22	*	$\oplus$	$\rightarrow$	29
23	%	$\otimes$	$\hookleftarrow$	29
24	*	$\otimes$	$\uparrow$	29
25	%	$\oplus$	$\rightarrow$	17

Обозначения: % — хеширование делением; * — хеширование умножением; $\oplus$ — комбинирование слов ключа сложением по модулю; $\otimes$ — комбинирование слов ключа сложением по «исключающему или»; $\rightarrow$ — разрешение коллизий методом цепочек; $\hookleftarrow$ — разрешение коллизий методом наложения пространства записей на пространство ссылок; $\uparrow$ — разрешение коллизий методом «открытой адресации».

4.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание используемых методов хеширования и разрешения коллизий;

– описание реализации применённых методов хеширования и разрешения коллизий в программе;

– исходные данные к построению таблицы и проведению поиска;

– результат работы программы — построенная хеш-таблица и найденные ключи.

4.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о поиске методом хеширования и границах его применения;

– знать основные варианты построения хеширующих функций и способы разрешения коллизий;

– владеть методами программной реализации поиска путём хеширования;

– уметь подбирать необходимые для метода хеширования константы.

Лабораторная работа 5
Создание абсолютного ассемблера

5.1. Общие сведения

Основное назначение транслятора ассемблера — перевод программы, написанной на языке ассемблера в машинный (объектный) код.

При трансляции программы ассемблер выполняет по крайней мере следующие действия:

– выполняет разбор входных строк программы (ЛР 3);

– преобразует мнемонические коды операций в их эквиваленты на машинном языке, используя, как правило, хеш-таблицу (ЛР 4);

– преобразует символические операнды (метки) в соответствующие им адреса в памяти машины;

– строит машинные команды в соответствующих форматах и размещает их в адресном пространстве машины;

– выдаёт объектную программу и листинг.

Абсолютный ассемблер относится к наиболее простым. Он выполняет трансляцию программы, которая будет загружена с конкретного — заданного при трансляции — адреса памяти и в дальнейшем не может быть перемещена в другую область памяти.

В большинстве случаев ассемблер строится по двухпросмотровой схеме. Во время первого просмотра осуществляются обработка директив ассемблера и назначение адресов памяти всем символическим операндам, во время второго — окончательная трансляция, выдача объектного кода и листинга.

Для абсолютного ассемблера очень важной является директива `START` или аналогичная ей, предназначенная для задания адреса начальной загрузки программы.

Адрес запуска программы совсем не обязательно совпадает с адресом начала загрузки. Обычно адрес запуска указывают в директиве `END`, которая завершает текст программы.

Объектный код в реальном ассемблере должен выдаваться в бинарной форме. Но в учебных целях в настоящей лабораторной работе необходимо формировать объектный код в виде символьного файла, содержащего записи шестнадцатеричных цифр. Записи объектного кода обычно имеют некоторый формат, обеспечивающий передачу необходимой управляющей информации загрузчику. В случае данной лабораторной работы формат таких записей можно упростить.

Листинг программы должен включать назначенные строкам программы адреса, полученные в результате трансляции коды команд и сами исходные строки программы. Адреса и коды команд необходимо давать в шестнадцатеричной форме.

5.2. Задание на лабораторную работу

Написать программу абсолютного ассемблера для гипотетической вычислительной машины. Система команд машины должна включать арифметические команды, поразрядные логические команды, команды проверки условий и переходов, команды пересылки, команды ввода-вывода. Мнемокоды команд, а также соответствующие им коды машинных команд студенты должны подобрать самостоятельно.

При разработке программы ассемблера студенты должны опираться на исходные данные к лабораторным работам 3, 4 и использовать их результаты.

В качестве тестовой программы использовать программу вычисления факториала заданного числа.

5.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание команд (функция и формат) машины;

– текст программы ассемблера;

– текст исходной тестовой программы на ассемблере;

– результат трансляции тестовой программы: листинг и объектный код.

5.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о назначении и алогритме работы абсолютного ассемблера;

– знать основные схемы ассемблирования;

– владеть методами выбора схем ассемблирования под конкретные задачи;

– уметь практически реализовывать программы ассемблеров.

Лабораторная работа 6
Ознакомление с генератором программ синтаксического разбора YACC

6.1. Общие сведения

Генератор YACC предназначен для построения программ синтаксического анализа входного потока информации, которые могут применяться для создания различного рода трансляторов с языков высокого уровня. В настоящее время этот генератор является неотъемлемой частью ОС UNIX.

Подробное описание генератора YACC и методов его использования с подробными примерами дано в учебном пособии [4]. Лабораторная работа строится на базе этого учебного пособия.

6.2. Задание на лабораторную работу

Необходимо выполнить доработку калькулятора, разработанного в главе 4 учебного пособия [4]. Тип доработки студент выбирает в соответствии со своим номером в журнале из следующего списка.

1, 13. Добавить операцию унарного минуса.

2, 14. Добавить операцию возведения в степень `^`.

3, 15. Добавить операцию взятия остатка от деления `%`.

4, 16. Добавить операцию вычисления факториала `!`.

5, 17. Добавить операцию `@`, такую, что `n@` соответствует сумме $1 + 2 + \ldots + n$.

6, 18. Добавить операцию `$`, такую, что `n$` соответствует вычислению числа Фибоначчи с номером n.

7, 19. Добавить операцию `:`, такую, что `n:m` соответствует числу всех размещений из n различных элементов по m:

$$A_n^m = \frac{n!}{(n-m)!}.$$

8, 20. Добавить операцию `;`, такую что `n;m` соответствует числу всех сочетаний из n различных элементов по m:

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

9, 21. Добавить квадратные скобки `[·]`, соответствующие взятию целой части результата выражения, заключённого в них.

10, 22. Добавить фигурные скобки {·}, соответствующие взятию дробной части результата выражения, заключённого в них.

11, 23. Добавить прямые скобки |·|, соответствующие вычислению модуля результата выражения, заключённого в них.

12, 24. Добавить угловые скобки <·>, соответствующие округлению до ближайшего целого результата выражения, заключённого в них.

6.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на доработку калькулятора;

– краткое описание доработки с обоснованием принятых решений;

– выражение для тестирования доработанной программы;

– результат обработки выражения.

6.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о назначении и методах использования генератора YACC;

– знать основные элементы языка спецификаций YACC;

– владеть методами построения программ синтаксического разбора с применением YACC;

– уметь применять генератор YACC при разработке прикладных и системных программ.

Лабораторная работа 7
Ознакомление с генератором программ лексического разбора LEX

7.1. Общие сведения

Генератор LEX предназначен для построения программ лексического анализа входного потока информации, которые могут применяться для создания различного рода трансляторов с языков высокого уровня, а также программ по обработке текстовых файлов. В настоящее время этот генератор является неотъемлемой частью ОС UNIX.

Подробное описание генератора LEX и методов его использования с примерами дано в учебном пособии [4], глава 3. Лабораторная работа строится на базе этого учебного пособия.

7.2. Задание на лабораторную работу

Написать на LEX программу в соответствии с заданием, соответствующим номеру студента в журнале.

1. Написать программу подсчёта числа предложений в файле. Окончанием каждого предложения считать символ точки.

2. Написать программу, удаляющую из файла пустые строки.

3. Написать программу, вставляющую пустую строку после каждой строки файла.

4. Написать программу, сжимающую повторяющиеся пробелы в один.

5. Написать программу, заменяющую символы табуляции на символ пробела.

6. Написать программу, заменяющую парные кавычки `"..."` на пары `<<...>>`.

7. Написать программу, выполняющую переформатирование файла так, чтобы строки были не длинее 70 символов и сохранялась целостность слов.

8. Написать программу, печатающую каждое слово из файла на отдельной строке.

9. Написать программу, удаляющую из текстового файла слова, состоящие из латинских букв.

10. Написать программу, удаляющую из текстового файла слова, состоящие из русских букв.

11. Написать программу, удаляющую из текстового файла слова, состоящие из цифр.

12. Написать программу, удаляющую из текстового файла слова, состоящие из небукв и нецифр.

13. Написать программу, подсчитывающую количество однобуквенных, двухбуквенных и трёхбуквенных слов в файле.

14. Написать программу, проверяющую, что каждое предложение в файле начинается с прописной буквы.

15. Написать программу, выполняющую удаление переносов слов из файла (выполняющую объединение слов).

16. Написать программу, обеспечивающую правильное, с точки зрения вёрстки, положение знаков препинания в тексте; например, пробел никогда не ставится перед запятой, но обязательно после неё и т. п.

17. Написать программу, обеспечивающую правильное, с точки зрения вёрстки, положение скобок в тексте.

18. Написать программу приведения всех слов файла к нижнему регистру.

19. Написать программу приведения всех слов файла к верхнему регистру.

20. Написать программу деления текстового файла на страницы заданного размера.

21. Написать программу, обеспечивающую распознавание типов чисел, содержащихся в текстовом файле.

22. Написать программу, обеспечивающую приведение всех чисел в текстовом файле к десятичной системе счисления.

23. Написать программу, обеспечивающую приведение всех чисел в текстовом файле к восьмеричной системе счисления.

24. Написать программу, обеспечивающую приведение всех чисел в текстовом файле к шестнадцатеричной системе счисления.

25. Написать программу, обеспечивающую приведение всех чисел в текстовом файле к двоичной системе счисления.

7.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на лабораторную работу;

– краткое описание решения задачи;

– текст программы, являющейся решением задачи, на LEX;

– тестовый пример;

– результат обработки тестового примера.

7.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о назначении и методах использования генератора LEX;

– знать основные элементы языка спецификаций LEX;

– владеть методами построения программ лексического разбора с применением LEX;

– уметь применять генератор LEX при разработке прикладных и системных программ.

Лабораторная работа 8
Сравнение реальной и расширенной машин

8.1. Общие сведения

При работе с операционными системами необходимо различать понятия реальной и расширенной машины. **Реальная машина** — набор аппаратных средств самой ЭВМ, предоставляющий, в частности, набор команд процессора, регистров ввода-вывода периферийных устройств и т. п. **Расширенная машина** — набор стандартных подпрограмм, предоставляемых операционной системой прикладным программам в качестве стандартных средств по выполнению различных системных функций (распределение памяти, организация ввода-вывода и др.). При этом средства расширенной машины значительно проще в использовании и менее подвержены ошибкам, поскольку в них учтено множество нюансов выполнения соответствующих операций.

В качестве конкретного примера реальной и расширенной машин рассмотрим подсистему вывода текстовой информации на терминал ПЭВМ IBM PC. Здесь реальная машина представлена аппаратурой видеоадаптера, имеющего свои регистры управления режимами работы и собственную память для хранения данных. Расширенная машина представлена прерываниями BIOS и DOS.

Описание аппаратных средств видеоадаптера

Управление режимами видеоадаптеров — весьма сложный процесс. Типичный видеоадаптер имеет несколько десятков управляющих регистров различного назначения. Причём ошибочная запись в некоторые из них может привести к порче монитора. В последнее время вышло довольно большое количество литературы, посвящённой программированию видеоадаптеров. В качестве примера можно указать книгу [6].

Здесь будет рассмотрен только цветной текстовый видеорежим разрешением 80×25 как наиболее совместимый для большинства видеоадаптеров. Как правило, этот режим устанавливается по умолчанию при загрузке компьютера. Поэтому специально программировать его нет смысла.

В этом режиме информация, отображаемая на экране терминала, размещается в видеопамяти, начиная с адреса B800:0000h. Каждый отображаемый на терминале символ занимает два последовательных

байта памяти: младший — код символа; старший — цвет символа. При этом младшая тетрада задаёт цвет собственно символа, старшая — цвет фона. Цвет кодируется по системе RGB следующим образом: самый младший бит — синий; следующий — зелёный; далее — красный. Самый старший бит в тетраде задаёт интенсивность цвета (обычный или яркий). При этом в тетраде, отвечающей за цвет фона, этот бит в зависимости от режима работы адаптера может управлять не интенсивностью, а миганием символа.

Предположим, что нам необходимо записать символ в шестую позицию пятой строки. Для этого необходимо вычислить адрес ячейки в видеопамяти, в которой должен располагаться символ для отображения в этой позиции.

Одна строка терминала занимает в видеопамяти

$$80 \text{ символов} \times 2 \text{ байта} = 160 \text{ байт}.$$

Следовательно, полное смещение от начала видеопамяти для нашего случая будет

$$160 \times (5 - 1) + 2 \times (6 - 1) = 650 = 28\text{Ah}.$$

То есть нам необходимо записать код символа по адресу B800:028Ah и код его цвета по адресу B800:028Bh.

Для вывода строки необходимо аналогичным образом заполнять последовательные байты видеопамяти. Если при этом не контролировать положение правой границы строки на терминале, то длинная строка может автоматически занять несколько терминальных строк.

Описание функций BIOS

Всё управление видеосервисом сведено в прерываение 10h. Здесь приводится описание только тех функций прерывания, которые могут понадобиться для выполнения лабораторной работы.

На первый взгляд может показаться, что использование функций BIOS более громоздко, чем программирование видеоадаптера напрямую. Однако использование BIOS обеспечивает значительно бóльшую переносимость программы между различными типами видеоадаптеров. Попытка достичь такой же переносимости прямым программированием видеоадаптера потребует весьма большого объёма кода и при этом не гарантирует результат.

Функция 02h — установка позиции курсора. Применяется для позиционирования курсора. Позиция задаётся относительно верхнего левого угла экрана, имеющего координаты (0,0).

Вход:

AH=02h

BH=номер страницы (обычно 0)

DH=номер строки

DL=номер столбца

Возврат: нет.

Функция 06h — скроллинг окна вверх. Выполняет «прокрутку» содержимого заданного окна экрана вверх на заданное число строк.

Вход:

AH=06h

AL=число строк для скроллинга

BH=атрибут для вставленных снизу строк

CH=номер строки верхнего левого угла окна

CL=номер столбца верхнего левого угла окна

DH=номер строки нижнего правого угла окна

DL=номер столбца нижнего правого угла окна

Возврат: нет.

Функция 07h — скроллинг окна вниз. Выполняет «прокрутку» содержимого заданного окна экрана вниз на заданное число строк.

Вход:

AH=07h

AL=число строк для скроллинга

BH=атрибут для вставленных сверху строк

CH=номер строки верхнего левого угла окна

CL=номер столбца верхнего левого угла окна

DH=номер строки нижнего правого угла окна

DL=номер столбца нижнего правого угла окна

Возврат: нет.

Функция 09h — запись символа и атрибута в позицию курсора. Записывает символ и атрибут заданное число раз, начиная с позиции курсора. Курсор не перемещается. Специальные символы (перевод строки и т. п.) отдельно не выделяются и выводятся, как и другие символы.

Вход:

AH=09h

AL=код символа

BH=страница (обычно 0)
BL=цвет символа и фона (атрибут)
CX=коэффициент повторения
Возврат: нет.

Функция 0Ah — запись символа в позицию курсора. Записывает символ заданное число раз, начиная с позиции курсора. Цветовой атрибут не изменяется. Курсор не перемещается. Специальные символы (перевод строки и т. п.) отдельно не выделяются и выводятся, как и другие символы.

Вход:
AH=0Ah
AL=код символа
BH=страница (обычно 0)
CX=коэффициент повторения
Возврат: нет.

Функция 0Eh — запись символа в позицию курсора в телетайпном режиме. Записывает символ в позицию курсора. Цветовой атрибут не изменяется. Курсор перемещается в следующую позицию. Распознаются и обрабатываются четыре управляющих символа: звонок — 07h; возврат на шаг — 08h; перевод строки — 0Ah; возврат каретки — 0Dh.

Вход:
AH=0Eh
AL=код символа
BH=страница (обычно 0)
Возврат: нет.

Функция 13h — запись цепочки символов (строки). Записывает в видеобуфер цепочку символов с указанными атрибутами. Распознаёт и обрабатывает четыре указанных выше управляющих символа.

Если атрибуты ожидаются в самой цепочке символов, то за каждым кодом символа должен следовать код его атрибута.

Вход:
AH=13h
AL=
00h — в BL атрибут, курсор не перемещается
01h — в BL атрибут, курсор перемещается
02h — атрибуты в цепочке, курсор не перемещается
03h — атрибуты в цепочке, курсор перемещается
BH=страница (обычно 0)

BL=атрибут
CX=длина цепочки
DH=номер строки экрана
DL=номер столбца экрана
ES:BP=начальный адрес цепочки
Возврат: нет.

Описание прерываний DOS

Описываемый сервис DOS ориентирован на вывод данных в стандартный поток вывода, который по умолчанию связан с экраном терминала. В связи с этим появляется возможность перенаправить весь вывод на другое устройство или в файл, практически не изменяя самой программы. Весь сервис представлен функциями прерывания 21h. Обратите внимание, что в сервисе DOS не предусмотрена работа с цветовыми атрибутами.

Функция 02h — выдать символ на стандартный вывод. Обрабатывает символ 08h (шаг назад). Распознаёт Ctrl Break и вызывает обработчик.

Вход:
AH=02h
DL=код символа
Возврат: нет.

Функция 06h* — выдать символ на стандартный вывод.

Вход:
AH=06h
DL=код символа от 0h до 0FEh
Возврат: нет.

Функция 09h — выдать строку на стандартный вывод. Строка должна быть ограничена символом '$' (24h). Распознаются и обрабатываются символы 08h (шаг назад), 0Ah (перевод строки), 0Dh (возврат каретки). Кроме того, опознаётся Ctrl Break и передаётся управление обработчику.

Вход:
AH=09h
DS:DX=адрес начала выводимой строки. Строка должна завершаться символом '$'.

---

* Помеченные таким знаком функции имеют более широкое назначение, чем описанное здесь.

Возврат: нет.

Функция 40h* — выдать строку на стандартный вывод. Осуществляет выдачу строки на стандартный вывод как запись в стандартный файл вывода. Строка ограничивается по длине.

Вход:

AH=40h

BX=описатель файла (в данном случае 1)

DS:DX=адрес начала выводимой строки

CX=число байт

Возврат:

AX=код ошибки, если установлен CF

AL=число реально выданных байт

8.2. Задание на лабораторную работу

Написать на ассемблере IBM PC программу вывода строки на экран:

– прямой записью в видеопамять;

– с использованием функции BIOS.

При этом руководствоваться требованиями к программе, представленными в табл. 8.1.

8.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание алгоритма работы с аппаратурой, а также используемых прерываний и функций;

– тексты программ;

8.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о понятиях реальной и расширенной машин;

– знать назначение расширенной машины IBM PC;

– владеть методами проектирования расширенной машины на базе реальной;

– уметь применять сервис расширенной машины IBM PC.

Таблица 8.1

Номер в журнале	Цветность символов	В цветности задан	Направление строки
1	Один цвет	Цвет символов	↑
2	Разные цвета	Цвет фона	↑
3	Один цвет	Оба цвета	→
4	Разные цвета	Цвет символов	→
5	Один цвет	Цвет фона	↓
6	Разные цвета	Оба цвета	↓
7	Один цвет	Цвет символов	←
8	Разные цвета	Цвет фона	←
9	Один цвет	Оба цвета	↑
10	Разные цвета	Цвет символов	↑
11	Один цвет	Цвет фона	→
12	Разные цвета	Оба цвета	→
13	Один цвет	Цвет символов	↓
14	Разные цвета	Цвет фона	↓
15	Один цвет	Оба цвета	←
16	Разные цвета	Цвет символов	←
17	Один цвет	Цвет фона	↑
18	Разные цвета	Оба цвета	↑
19	Один цвет	Цвет символов	→
20	Разные цвета	Цвет фона	→
21	Один цвет	Оба цвета	↓
22	Разные цвета	Цвет символов	↓
23	Один цвет	Цвет фона	←
24	Разные цвета	Оба цвета	←
25	Один цвет	Цвет символов	↑

Лабораторная работа 9
Методы распределения оперативной памяти

9.1. Общие сведения

Любая современная операционная система обеспечивает возможность распределения оперативной памяти между процессами с помощью двух системных вызовов: «выделить блок памяти» и «освободить блок памяти». При этом предполагается, что для выделения может быть запрошен блок памяти произвольного размера*, а освобождён только ранее выделенный блок памяти целиком.

Стратегии выделения памяти

Запрошенный блок памяти выделяется операционной системой из имеющейся в наличии непрерывной свободной оперативной памяти. Как правило, при устоявшемся режиме работы системы свободная память представляет собой набор «дыр» некоторого размера среди блоков уже распределённой памяти. Таким образом, запрошенный блок должен по размеру быть не больше хотя бы одной «дыры».

Если в настоящий момент блок выделить невозможно, система может принять некоторые специальные меры: заблокировать запросивший память процесс до момента, пока станет доступной «дыра» подходящего размера; освободить несколько блоков оперативной памяти (с применением свопинга) для получения «дыры» подходящего размера; выполнить процедуру уплотнения памяти перемещением всех распределённых блоков в единую сплошную область со стремлением создать достаточно большую «дыру».

Рассмотрим только простейший случай, при котором запрошенный блок памяти выбирается из множества имеющихся «дыр». Если подходящая «дыра» не найдена, то процедура выделения памяти возвращает признак отказа.

Поиск подходящей «дыры» может быть реализован по одной из двух стратегий: «первый подходящий» и «наилучший подходящий».

При использовании стратегии «первый подходящий» ищут любую «дыру», такую, чтобы она была не меньше размеров запрошенного блока. При стратегии «наилучший подходящий» выбирают ту

* На практике запросы блоков памяти, превышающих физический размер оперативной памяти, бессмысленны.

«дыру», которая подходит по размерам наилучшим образом, то есть по размеру больше запрошенного блока на наименьшую величину из всех существующих «дыр».

Память может быть распределена как часть найденной «дыры» с уменьшением размеров самой «дыры» или как «дыра» целиком, если остающаяся нераспределённой часть «дыры» мала для того, чтобы иметь какую-либо ценность в будущем.

Стратегия «первый подходящий» работает быстрее, но преждевременно распределяет большие «дыры». Стратегия «наилучший подходящий» сохраняет большие дыры, но медленнее и имеет тенденцию к созданию большого числа маленьких «дыр», которые скорее всего не смогут быть использованы в дальнейшем.

При использовании стратегии «первый подходящий» наблюдается тенденция к образованию большого количества маленьких «дыр» у начала списка свободных блоков, что в конечном итоге увеличивает время поиска подходящей «дыры». Этого можно избежать, начиная поиск каждый раз не в одном и том же месте, а в случайно выбранной точке списка, имея возможность при этом просмотреть в конечном итоге весь список.

Обе стратегии имеют как положительные, так и отрицательные стороны. При определённых условиях любая из двух стратегий может превзойти другую. Выбор конкретной стратегии на практике связан с анализом результатов моделирования поведения задач в конкретной системе.

Освобождение памяти

Когда блок памяти становится не нужен, он возвращается операционной системе, и в памяти появляется новая «дыра». При этом, если эта «дыра» оказывается смежной с уже существующей «дырой», то необходимо объединить две «дыры» в одну «дыру» бóльшего размера. Эта задача может оказаться достаточно сложной на практике.

Предположим, что список «дыр» хранится в виде простого односвязного списка (рис. 9.1,*а*). При освобождении памяти наиболее простым является добавление информации о новых «дырах» к одному из концов списка (обычно к началу). Поэтому, если в начальный момент времени список был упорядочен по возрастанию адресов, то с течением времени упорядоченность будет естественным образом нарушена. Можно, конечно, принять специальные меры по поддержанию упорядоченности списка (рис. 9.1,*б*), но это потребует проведения поиска в списке места, куда должен быть помещён освобождённый блок,

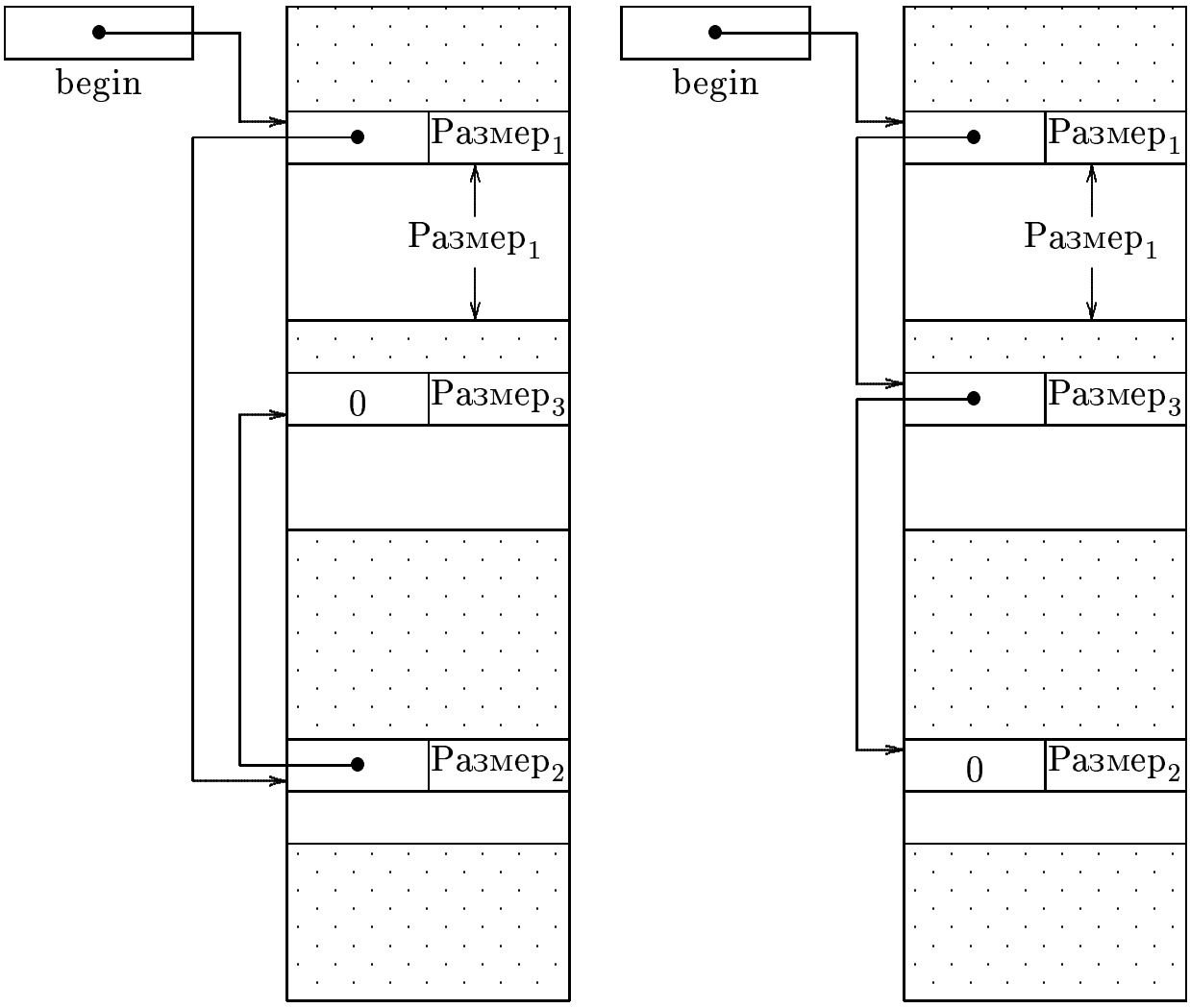

Рис. 9.1. Хранение информации о «дырах» в виде односвязного списка: *а* — неупорядоченного; *б* — упорядоченного по возрастанию адресов памяти; заштрихованы уже распределённые блоки памяти

то есть дополнительных накладных расходов.

Для эффективного решения задачи объединения «дыр» должны использоваться соответствующие структуры данных.

Структуры данных для управления распределением памяти

В простейшем случае память может быть поделена на блоки фиксированного (небольшого) размера. В этом случае в ответ на запрос выделяется непрерывная последовательность блоков, достаточная по суммарному объёму. Для хранения информации о занятых и свободных блоках памяти используется так называемая битовая карта памяти, в которой каждому биту соответствует свой блок памяти. Если блок занят, бит равен единице, иначе — нулю.

В более общем случае используется выделение блоков памяти произвольного размера, соответствующих запросу. При этом информацию о свободных блоках памяти хранят в самих блоках, размещая её как запись в начале каждого свободного блока («дыры») и организуя эти записи в виде списка (см. рис. 9.1). Каждая запись содержит информацию о размере блока и указатель на следующий блок.

Как уже было показано, при такой организации структур данных для определения сопряжённых «дыр» удобнее всего поддерживать список блоков упорядоченным по адресам памяти, то есть добавлять освобождаемые блоки не в какой-либо конец списка, а в середину, в соответствии с их адресами.

Усложнив структуры данных, можно избавиться от упорядочивания по адресам. При этом создаётся двусвязный список свободных блоков, а каждый блок в начале и в конце снабжается специальными «этикетками», которые содержат информацию о том занят блок или свободен (рис. 9.2). При освобождении блок может быть записан в любое место списка. Проверка на сопряжённые «дыры» выполняется по расположенным в соседних ячейках памяти «этикеткам».

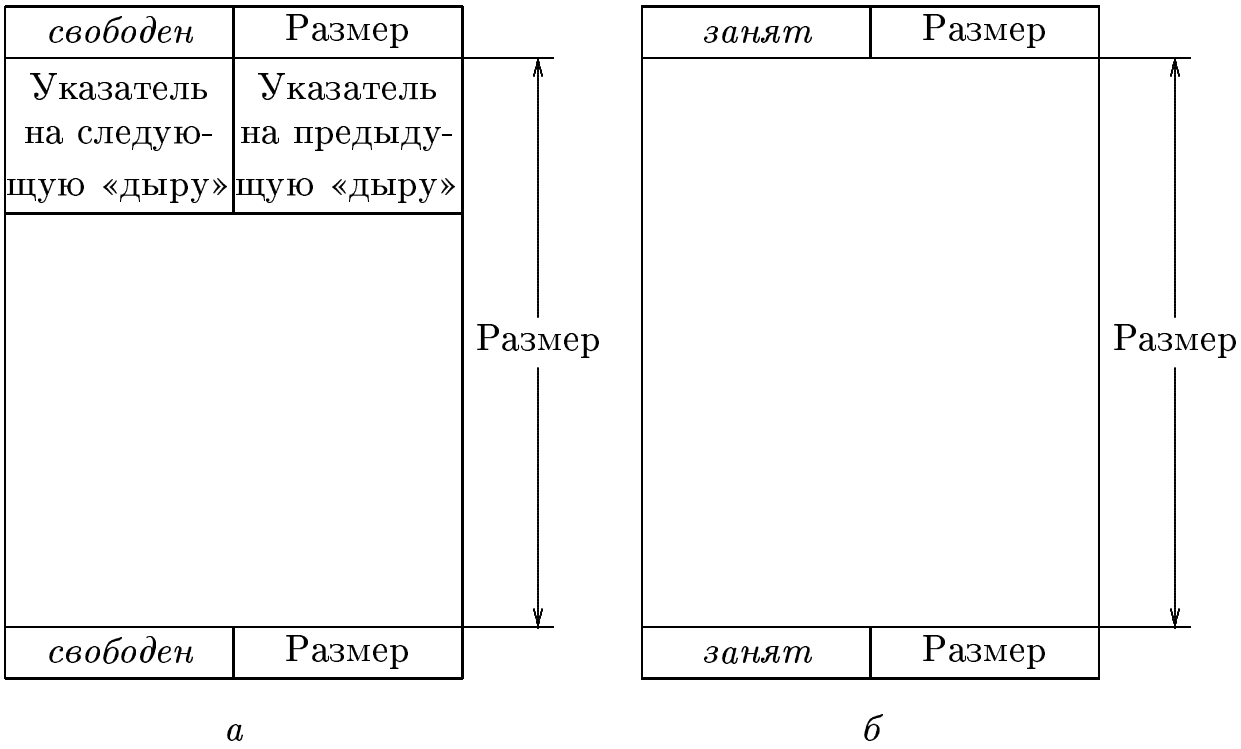

Рис. 9.2. Блок данных — элемент двусвязного списка с «этикетками»: *а* — свободный («дыра»); *б* — распределённый (занятый)

Подробнее о структурах данных и стратегиях распределения памяти см. [8], с. 163 – 171.

9.2. Задание на лабораторную работу

Написать на языке Си программу, демонстрирующую один из способов распределения памяти, выбираемый студентом из табл. 9.1 в соответствии со своим номером в журнале. Программа должна в начале работы запросить максимальный объём памяти, а затем применять собственные (разработанные студентом) процедуры распределения памяти для выделения и освобождения блоков памяти. Процедурами распределения памяти считаются процедуры выделения блока памяти заданного размера и освобождения ранее выделенного блока памяти. Процедура выделения блока памяти должна возвращать спецификатор начала выделенного блока. Процедура освобождения блока памяти должна освобождать блок, заданный спецификатором. Результаты работы программы должны быть представлены в печатной форме и внесены в отчёт.

Таблица 9.1

Номер по журналу	Стратегия размещения	Структура данных
1, 2, 3, 4	«Первый подходящий»	Блоки фиксированного размера
5, 6, 7, 8	«Наилучший подходящий»	Односвязный упорядоченный список
9, 10, 11, 12	«Первый подходящий»	Двусвязный список с этикетками
13, 14, 15, 16	«Наилучший подходящий»	Блоки фиксированного размера
17, 18, 19, 20	«Первый подходящий»	Односвязный упорядоченный список
21, 22, 23, 24, 25	«Наилучший подходящий»	Двусвязный список с этикетками

9.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание используемых алгоритмов распределения памяти;

– исходный текст процедур управления памятью и демонстрационной программы;

– результаты работы демонстрационной программы.

9.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о способах распределения памяти, применяемых в вычислительных системах;

– знать основные структуры данных для управления памятью и стратегии распределения памяти;

– владеть методами выбора необходимых структур и стратегий; – уметь практически реализовать процедуры управления памятью.

Лабораторная работа 10
Подготовка гипертекстовых документов на HTML

10.1. Общие сведения

HTML — язык гипертекстовой разметки документов, предназначенный для создания документов, переносимых между различными платформами. При этом не гарантируется идентичность внешнего вида документа на разных платформах, но гарантируется сохранение логической структуры размеченного текста. То есть таблицы везде будут таблицами, заголовки — заголовками и т. п. В связи с этим HTML документы активно используются в Internet технологии WWW.

Полученный документ может быть просмотрен специальной программой-клиентом WWW, получившей название Browser*. Можно сказать, что клиент WWW интерпретирует документ. В связи с этим сам документ, размеченный в HTML, можно рассматривать как программу для получения печатного документа.

Для изучения HTML отошлём читателя к справочной литературе, которой в настоящее время вышло достаточно много. В качестве примера укажем книгу [7]. Здесь же рассмотрим только основы HTML.

Язык представляет собой набор тегов, вставляемых в текст документа и обеспечивающих тем самым его разметку. Теги представляют собой конструкции вида `<слово>`, где `слово` — некоторое ключевое слово. Обычно теги образуют пары из открывающего и закрывающего. Закрывающий тег содержит в своём теле дополнительный символ слеша `</слово>`. Некоторые теги закрывающих пар не имеют.

Стандартом HTML предусмотрено, что если клиент WWW встречает тег, который не может интерпретировать, то он его просто пропускает.

Общая структура документа на HTML выглядит следующим образом:

```
<HTML>
<HEAD>
<TITLE>Заголовок окна</TITLE>
</HEAD>
```

* Автор затрудняется указать русский аналог этого термина. Поэтому в дальнейшем будет использовать термин «клиент WWW».

```
<BODY>
Тело документа
</BODY>
</HTML>
```

Из приведённой структуры видно, что документ обычно содержит две части: заголовок, образованный парой тегов `HEAD`, и собственно тело, образованное парой тегов `BODY`. Заголовок по стандарту является обязательной частью документа, но содержит лишь служебную информацию. В частности, он содержит текстовую строку, которая должна быть напечатана в шапке окна клиента WWW, отображающего документ.

В теле документа весь текст, не содержащий тегов разметки, считается одним абзацем и соответствующим образом форматируется программами-клиентами WWW. При этом игнорируются все множественные пробелы и возвраты каретки.

Из тегов разметки тела документа укажем только наиболее употребительные.

Парные теги:

`<CENTER></CENTER>` — центрировать текст;

`<H1></H1>` — заголовок первого уровня (есть уровни до 6);

`<B></B>` — полужирный шрифт;

`<I></I>` — курсив;

`<A></A>` — гипертекстовая ссылка в документе или на другой документ;

`<PRE></PRE>` — не форматировать текст между тегами.

Непарные теги:

`<P>` — начать новый абзац;

`<BR>` — разорвать строку, начать с новой строки;

`<HR>` — провести горизонтальную линию;

`<IMG>` — вставка картинки.

Картинки могут быть в форматах GIF и JPEG. В последнее время начинает использоваться формат PNG.

HTML чрезвычайно удобно использовать для автоматического форматирования документов, полученных как отчёты при статистической обработке информации из файлов log'ов или при выборке из базы данных.

10.2. Задание на лабораторную работу

Подготовить на HTML несколько документов для своей домашней WWW странички. Документы должны включать в себя картинки и гипертекстовые ссылки внутри документов и между ними.

10.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание используемых тегов HTML;

– HTML код размеченных документов;

– распечатку внешнего вида просмотренных документов.

10.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о назначении языка разметки HTML;

– знать основные теги HTML;

– владеть методами деления больших документов на логически завершённые части;

– уметь выполнять разметку HTML документов.

Лабораторная работа 11
Подготовка печатных документов с помощью системы TEX

11.1. Общие сведения

В отличие от HTML, TEX представляет собой универсальную систему вёрстки документов. Он не предназначен для подготовки гипертекстовых документов. Его назначение — подготовка полиграфически грамотно набранных документов для печати в типографии. TEX специально спроектирован для подготовки документов, содержащих сложные математические формулы. В возможностях вёрстки математики на сегодняшний день ему нет равных.

В настоящее время TEX реализован практически на всех аппаратных платформах для всех операционных систем (ОС). Поэтому можно **гарантировать**, что документ, подготовленный в одной ОС на данной платформе, будет **абсолютно идентично** воспроизведён на любой другой платформе в любой другой ОС*. Такой переносимостью не может на сегодня похвастаться ни одна другая система вёрстки документов. Поэтому многие научные издательства в мире предпочитают получать документы для печати, подготовленные именно в TEX'е. Во многих научных обществах, в том числе и в нашей стране, TEX выбран в качестве внутреннего стандарта для обмена научными документами и статьями. В нашей стране также есть издательства, выполняющие вёрстку в TEX'е. Для примера укажем широко известное московское издательство «Мир».

Такое широкое распространение TEX'а объясняется и тем, что он носит статус public domain, то есть распространяется свободно (бесплатно) в учебных и просветительских целях**.

---

* Это заявление относится к документам, подготовленным на языках, использующих латиницу. Для русскоязычных документов существует проблема нескольких несовместимых между собой по кодировкам русификаций. Однако она может быть решена с использованием программ-конвертеров из одной кодировки в другую и передачей документа в исходном виде. Работа в направлении стандартизации русификаций, к сожалению, не ведётся.

** Существуют и коммерческие реализации TEX'а.

Принцип подготовки документов на TEX'е тот же, что и на HTML. Текст документа размечается командами, которые представляют собой обычные слова, записываемые с обратным слешом в начале (`\команда`). Далее размеченный документ транслируется в так называемый независимый от устройства формат (device independent — dvi). Этот формат представляет собой уже сформатированный документ и может быть воспроизведён без каких-либо изменений на устройстве печати любого разрешения (качество внешнего вида будет, конечно, зависеть от разрешения устройства, но расположение элементов документа будет одним и тем же на любом устройстве).

Исходный документ может быть подготовлен в любом текстовом редакторе, дающем на выходе простой ASCII текст, или даже получен методом автоматической генерации, что позволяет применять TEX как составную часть систем автоматизированной подготовки документов.

Команды TEX'а делятся на примитивы, встроенные в саму систему, и макрокоманды, определённые в рамках форматов и макропакетов расширения. На сегодняшний день существует три наиболее распространённых формата: plain, LATEX и $\mathcal{AMS}$-TEX, а также большое количество дополнительных макропакетов (в качестве примера назовём макропакеты PICTEX для подготовки рисунков и XY-pic для подготовки диаграмм).

Рассмотрим здесь только наиболее простые команды plain TEX'а. Заинтересовавшихся системой отправим к книгам [9], [10].

Прежде всего приведём список символов, являющихся служебными. Их нельзя применять в простом тексте без специальных команд:
`\ { } $ & # ^ _ % ~`
Если эти символы должны встретиться в тексте, то вводить их надо так: `\backslash`, `\{`, `\}`, `\$`, `\&`, `\#`, `\^`, `\_`, `\%`, `\~`.

Текст абзаца набирается обычным образом без переносов с произвольным числом пробелов между словами, произвольным отступом от левого края, без отступа красной строки. Длина строки произвольная. При трансляции TEX сам выполнит форматирование абзаца в соответствии с параметрами используемого формата. При необходимости переносы слов будут вставлены автоматически.

Абзацы между собой отделяются произвольным количеством пустых строк.

Если в текст необходимо вставить комментарий, который не должен попасть в итоговый документ, используется символ `%`. Весь текст, начиная от этого символа и до конца строки, при форматировании

документа игнорируется.

При подготовке документа можно использовать следующие команды переключения шрифтов:

`\sl` — *наклонный шрифт*;

`\bf` — **полужирный шрифт**;

`\it` — *курсивный шрифт*;

`\tt` — `равноширинный шрифт (пишущая машинка)`;

`\rm` — обычный шрифт (включён по умолчанию).

Каждая команда включает соответствующий шрифт, отменяя действие предыдущей. Комбинировать команды, скажем `\bf\it` для получения ***полужирного курсива***, нельзя.

Символы `{` и `}` используются для группирования текста. Действие команд TEX'а, взятых в группы, отменяется за пределами этих групп. Группирование также может использоваться для задания аргументов макрокоманды.

Можно, например, так набрать абзац, использующий несколько шрифтов:

```
Текст абзаца содержит обычный шрифт, {\bf полужирный шрифт} и
{\it курсивный шрифт}.
```

В результате получим:

Текст абзаца содержит обычный шрифт, **полужирный шрифт** и *курсивный шрифт*.

По умолчанию текст форматируется в абзацы с абзацным отступом в красной строке и выровненными левым и правым краями. Последняя строка абзаца имеет недозаполнение. Если набрать короткую строку, то окажется, что она будет выровнена влево, но с абзацным отступом.

Выровнять (одну) строку произвольным образом можно макрокомандами:

`\leftline{строка}` — прижимает строку к левому краю;

`\centerline{строка}` — центрирует строку;

`\rightline{строка}` — прижимает строку к правому краю.

Макрокоманды можно комбинировать. Например:

```
\leftline{\bf Первая строка. }
\centerline{\it Вторая строка. }
\rightline{\sl Третья строка. }
```

Даст при печати:

Первая строка.

Вторая строка.

Третья строка.

При необходимости оставить между абзацами текста некоторый пробел используются макрокоманды:

`\smallskip` — пропуск в полстроки;

`\medskip` — пропуск в одну строку;

`\bigskip` — пропуск в полторы строки.

Набор математических формул в TEX'е на первый взгляд может показаться достаточно сложным. Однако на самом деле он базируется на простых принципах. Освоить его не сложнее, чем в любом другом текстовом процессоре.

Запись формул выполняется в специальных «скобках» — `$`, если формула включена в текст абзаца, и `$$`, если формула выделенная. Формулы записываются в том порядке, как читаются. При записи индексов используются символы: `_` — для записи нижнего индекса; `^` — для записи верхнего индекса. Если индексы состоят не из одного символа, они должны быть взяты в фигурные скобки `{` и `}`.

Приведём пример:

```
Число всех перестановок из $n$ различных элементов (обозначается
$P_n$): $$P_n=1\cdot 2\cdot 3\cdot\ldots\cdot n=n!=A_n^n.$$
```

Даст при печати:

Число всех перестановок из n различных элементов (обозначается P_n):

$$P_n = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n! = A_n^n.$$

Как можно видеть из примера, для набора символов, отсутствующих на клавиатуре, используются специальные макрокоманды. Существует довольно большой список макрокоманд, дающих различные математические символы. Приведём некоторые из них для примера:

`\pm` $\pm$ `\times` $\times$ `\div` $\div$ `\vee` $\vee$ `\circ` $\circ$ `\to` $\to$
`\cdot` $\cdot$ `\ast` $\ast$ `\in` $\in$ `\wedge` $\wedge$ `bullet` $\bullet$ `\infty` ∞

Греческие буквы набирают по их названиям: `\alpha` даёт α, `\beta` — β и т. д.

Существуют также макрокоманды для записи стандартных математических функций (приведены только некоторые для примера) и больших операторов:

`\sin`	sin	`\lim`	lim	`\sum`	$\sum$	`\int`	$\int$	`\bigvee`	$\bigvee$
`\cos`	cos	`\log`	log	`\prod`	$\prod$	`\oint`	$\oint$	`\bigwedge`	$\bigwedge$

К большим операторам так же, как и к математическим функциям, могут быть приписаны необходимые пределы и индексы при помощи уже описанных символов `_` и `^`.

Описанные выше символы, греческие буквы, математические функции и большие операторы могут быть набраны только внутри математических скобок.

Примеры:

```
$$x_1+x_2+\ldots+x_n=\sum_{i=1}^n x_i=-a_1.$$
```

даёт:

$$x_1+x_2+\ldots+x_n=\sum_{i=1}^n x_i=-a_1.$$

```
$$x=r\sin\theta\cos\varphi.$$
```

даёт:

$$x=r\sin\theta\cos\varphi.$$

Для вёрстки дробей используется оператор `\over`. Его действие распространяется на всю формулу, поэтому необходимо ограничивать его действие фигурными скобками. Операция квадратного корня вводится оператором `\sqrt`. Пример:

```
$$Sy'^2+2 {ad+bc\over \sqrt{a^2+b^2}} y'+f=0.$$
```

даёт:

$$Sy'^2+2 {ad+bc\over \sqrt{a^2+b^2}} y'+f=0.$$

Для вёрстки матриц используется несколько макрокоманд, самая используемая из которых `\matrix`. Эта макрокоманда обеспечивает только размещение в требуемом порядке элементов матрицы, но не формирует вокруг матрицы никаких скобок. Элементы матрицы записываются построчно и разделяются в строке символами `&`. Строки матрицы разделяются комбинацией `\cr`.

Для создания скобок используется способность TEX'а к растягиванию стандартных скобок вокруг заданного объекта. Растягиваемые парные скобки задаются парными командами `\left` и `\right`, между

которыми помещается объект, вокруг которого должны быть размещены растянутые скобки.

Применение описанных конструкций проще всего продемонстрировать на примере:

```
$$A=\left( \matrix{
a_{11} & a_{12} & a_{13} \cr
a_{21} & a_{22} & a_{23} \cr
a_{31} & a_{32} & a_{33} \cr
} \right) .$$
```

даёт:

$$A = \left(\matrix{ a_{11} & a_{12} & a_{13} \cr a_{21} & a_{22} & a_{23} \cr a_{31} & a_{32} & a_{33} \cr } \right) .$$

Сами скобки можно выбирать из достаточно большого ряда. Приведём некоторые наиболее употребительные.

```
( (  [ [  \{ {  |  |  /          /  \langle ⟨  \lfloor ⌊  \lceil ⌈
) )  ] ]  \} }  \| ‖  \backslash \  \rangle ⟩  \rfloor ⌋  \rceil ⌉
```

В заключение рассмотрим пример подготовки на TEX'е титульного листа для отчёта о лабораторной работе. При разметке текста будет использовано несколько не описанных в тексте лабораторной работы команд, смысл которых понятен из результата форматирования.

```
\centerline{Московский государственный университет леса}
\centerline{Кафедра вычислительной техники}
\hrule

\vskip6cm
\centerline{\bf ОТЧЁТ}
\centerline{по лабораторной работе \No 1}
\medskip
\centerline{ПРИМЕНЕНИЕ ФИЛЬТРОВ ОС UNIX}
\vskip2cm
\rightline{Выполнил студент гр. ВТ-31 И.И.~Иванов}
\bigskip
\rightline{Преподаватель: доцент А.В.~Чернышов}

\vfill
\centerline{Москва --- 2000}
```

Результат форматирования представлен на рис. 11.1.

На этом закончим краткое введение в plain TEX. За дополнительной информацией отошлём читателя к литературе.

11.2. Задание на лабораторную работу

Каждому студенту подготовить на TEX'е документ с предложениями по совершенствованию учебной программы и по улучшению учебного процесса в нашем университете. Документ должен содержать 2...3 страницы текста и включать:

– заголовок;
– анализ сложившейся ситуации;
– 2...4 конкретных предложения.

Или выполнить набор и вёрстку одной страницы любого справочника по математике. При этом выбрать страницу, номер которой оканчивается на номер студента в журнале.

11.3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;
– задание на работу;
– краткое описание используемых конструкций языка TEX'а;
– исходный текст размеченного документа;
– результаты форматирования (трансляции) документа.

11.4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о назначении системы TEX и её основных областях применения;

– знать основные команды системы TEX;

– владеть навыками разметки текста и математических формул в системе TEX;

– уметь применять систему TEX для подготовки несложных научных документов — рефератов, статей и т. п.

Московский государственный университет леса
Кафедра вычислительной техники

ОТЧЁТ
по лабораторной работе 1
ПРИМЕНЕНИЕ ФИЛЬТРОВ ОС UNIX

Выполнил студент гр. ВТ-31 И.И. Иванов

Преподаватель: доцент А.В. Чернышов

Москва — 2000

Рис. 11.1. Итоговый вид титульного листа

Литература

1. **Керниган Б. В., Пайк Р.** UNIX — универсальная среда программирования / Перевод с англ.; Предисл. М. И. Белякова. — М.: Финансы и статистика, 1992. — 304 с.

2. **Тихомиров В. П., Давидов М. И.** Операционная система ДЕМОС: Инструментальные средства программиста. — М.: Финансы и статистика, 1988. — 206 с.

3. **Беляков М. И., Рабовер Ю. И., Фридман А. Л.** Мобильная операционная система: Справочник. — М.: Радио и связь, 1991. — 208 с.

4. **Чернышов А. В.** Инструментальные средства программирования из состава ОС UNIX и их применение в повседневной практике: Учебное пособие для студентов специальности 2201. — М.: МГУЛ, 1999. — 191 с.

5. **Кнут Д. Е.** Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск / Перевод с англ. Н. И. Вьюковой, В. А. Галатенко, А. В. Ходулёва; Под ред. Ю. М. Баяковского и В. С. Штаркмана. — М.: Мир, 1978. — 844 с.

6. **Григорьев В. Л.** Видеосистемы ПК фирмы IBM. — М.: Радио и связь, 1993. — 192 с.

7. **Муллен Р.** HTML 4: справочник программиста / Перевод с англ. Ю. Стоцкого. — СПб.: Питер Ком, 1998. — 304 с.

8. **Шоу А.** Логическое проектирование операционных систем / Перевод с англ. В. В. Макарова и В. Д. Никитина; Под ред. Г. Н. Соловьёва. — М.: Мир, 1981. — 360 с.

9. **Кнут Д. Е.** Всё про TEX / Перевод с англ. М. В. Лисиной. — Протвино: АО RDTEX, 1993. — 592 с.

10. **Львовский С. М.** Набор и вёрстка в пакете LATEX. М.: Космосинформ, 1994. — 328 с.

11. **Бронштейн И. Н., Семендяев К. А.** Справочник по математике для инженеров и учащихся втузов. — М.: Наука, 1965. — 608 с.

Содержание