Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»

Кафедра электронных приборов

Информатика

КОМПЬЮТЕРНЫЙ ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Методические указания к лабораторным работам
для студентов направлений "Фотоника и оптоинформатика"
и «Электроника и микроэлектроника»
(специальность «Электронные приборы и устройства»

2011

Шандаров, Евгений Станиславович

Компьютерный лабораторный практикум = Информатика: методические указания к лабораторным работам для студентов направлений «Фотоника и оптоинформатика» и «Электроника и микроэлектроника» (специальность "Электронные приборы и устройства") / Е.С. Шандаров; Министерство образования и науки Российской Федерации, Государственное образовательное учреждение высшего профессионального образования Томский государственный университет систем управления и радиоэлектроники, Кафедра электронных приборов. - Томск : ТУСУР, 2011. - 85 с.

Предназначено для студентов очной и заочной форм, обучающихся по направлениям «Фотоника и оптоинформатика» и "Электроника и микроэлектроника" (специальность "Электронные приборы и устройства") по курсу «Информатика».

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»

Кафедра электронных приборов

УТВЕРЖДАЮ
Зав.кафедрой ЭП
_____С.М. Шандаров
«___» _____ 2011 г.

Информатика

КОМПЬЮТЕРНЫЙ ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Методические указания к лабораторным работам
для студентов направлений «Фотоника и оптоинформатика»
и «Электроника и микроэлектроника»
(специальность «Электронные приборы»)

Разработчик

ст. преподаватель каф.ЭП
_____Е.С. Шандаров
«____»_____2011 г

2011

Содержание

Раздел 1. Linux

Linux – свободно распространяемая многозадачная, многопользовательская операционная система.

Ядро Linux разработано Линусом Торвальдсом в 1991 г. Файлы первая версия Linux (версия 0.01) были опубликованы в Интернете 17 сентября 1991 года.

К основным характеристикам Linux можно отнести многозадачность, многопользовательский доступ и разграничение прав доступа к файлам, поддержка различных форматов файловых систем.

В мире существует уже более сотни различных дистрибутивов Linux. Наиболее широко распространенные: Ubuntu, OpenSuse, Fedora Core, Debian, Mandriva, Gentoo. Подавляющее большинство дистрибутивов распространяется по лицензии GPL.

Лабораторная работа №1. Работа с командной строкой Linux

1.1 Цель работы

Знакомство с командной строкой Linux, изучение основных команд (программ) Linux. Создание структуры файлов и каталогов.

1.2 Теоретическая часть

Вывод справки по команде (man)

`man <имя изучаемой команды>`

Команда Unix, предназначенная для форматирования и вывода справочных страниц.

Имя текущего пользователя (whoami)

whoami

Выводит действительный идентификатор пользователя

Список работающих процессов (top)

`top`

Показывает список работающих в данный момент процессов и информацию о них, включая использование ими памяти и процессора. Список интерактивно формируется в реальном времени.

Чтобы выйти из программы top, нажмите клавишу [q].

Отчет о работающих процессах (ps)

`ps [опции]`

Выводит в стандартный вывод информацию о текущем состоянии процессов. Опции

– `a` все терминальные процессы;

– `e` все процессы;

– `g` *список* выбирать процессы по *списку* лидеров групп;

– `p` *список* выбирать процессы по *списку* идентификаторов процессов;

– `t` *список* выбирать процессы по *списку* терминалов;

– `u` *список* выбирать процессы по *списку* идентификаторов пользователей;

– `f` генерировать полный листинг;

– `l` генерировать листинг в длинном формате.

Количество памяти (free)

`free [-b | -k | -m] [-o] [-s` *delay* `] [-t] [-V]`

Показывает общее количество свободной и используемой физической памяти и памяти отведенной для свопирования в системе, так же и совместно используемую память и буфера используемые ядром.

Опции:

`-b` показывает количество памяти в байтах; опция

– k (по умолчанию) показывает количество пвмяти в килобайтах;

– m показывает количество памяти в мегабайтах;

`-t` показывает строки содержащие полное количество памяти;

`-o` запрещает показывать строки относящиеся к "массиву буфера". Если не определено отнять/добавить память буферов из/в используемую/свободнуб память (соответственно!);

`-s` разрешает безостановочно выводить информацию с промежутком в *delay* секунд;

`-V` показывает информацию о версии программы.

Отчёт об использовании дискового пространства (df)

`df [`*опции*`] [`*файл...*`]`

Опции: `[-ahHiklmPv] [-t` *тип-файловой-системы*`] [-x` *тип-файловой-системы*`] [--block-size=`*размер*`] [--print-type] [--no-sync] [--sync] [--help] [--version] [--]`

Системная дата и время (date)

`date [`*ПАРАМЕТР*`]... [`*+ФОРМАТ*`]`

`date [`*-u*`|`*--utc*`|`*--universal*`] [`*ММДДччмм*`[[`*ВВ*`]`*ГГ*`][.`*сс*`]]`

Показывает текущее время в заданном ФОРМАТЕ, или устанавливает системную дату и время.

`-d, --date=`*`СТРОКА`* показывает время, описанное СТРОКОЙ – не 'текущее' время;

`-f, --file=`*`ФАЙЛ_ДАТ`* как и --date, только для каждой строки в ФАЙЛЕ_ДАТ ;

`-r, --reference=`*`ФАЙЛ`* показывает время последнего изменения ФАЙЛА ;

`-s, --set=`*`СТРОКА`* устанавливает время, заданное СТРОКОЙ;

`-u, --utc, --universal` выдает или устанавливает время по Гринвичу.

Параметр ФОРМАТ управляет исходящей информацией. Параметр работает только с временем по Гринвичу. Существуют такие интерпретируемые значения:

`%A` полное название дня недели в локали, с непостоянной длиной названия (Воскресенье..Суббота);

`%b` сокращенное название месяца в локали (Янв..Дек);

`%B` полное название месяца в локали, с непостоянной длиной названия (Январь..Декабрь);

`%c` дата и время в локали (Суб Ноя 04 12:02:33 EST 1989);

`%d` день месяца (01..31);

`%D` дата (мм/дд/гг);

`%e` день месяца, без нулей слева (1..31);

`%j` день года (001..366);

`%k` час (0..23);

`%l` час (1..12);

`%m` месяц (01..12);

`%M` минуты (00..59);

`%n` новая строка;

`%p` в зависимости от локали AM или PM (до полудня – после полудня);

`%t` горизонтальная табуляция;

`%V` номер недели в году с Понедельником, как первым днем недели (01..53);

`%w` номер дня в неделе (0..6); 0 принимается за Воскресенье;

`%W` номер недели в году с Понедельником, как первым днем недели (00..53);

`%x` представление даты в локали (мм/дд/гг);

`%X` представление времени в локали (%Ч:%М:%С);

`%Y` год (1970...).

По умолчанию, date заполняет пустые цифровые поля нулями. В GNU date распознает следующие модификаторы между `%' и цифровой командой:

-' (дефис) не заполнять поле `_' (подчеркивание) заполнить поле пробелами

Время работы системы (uptime)

```
uptime
uptime [-V]
```

uptime – показывает время работы системы с последнего запуска.

Выдача имени текущего каталога (pwd)

```
pwd
```

Выводит полное маршрутное имя текущего каталога

Смена текущего каталога (cd)

```
cd [каталог]
```

Команда cd применяется для того, чтобы сделать заданный каталог текущим. Если каталог не указан, используется значение переменной окружения $HOME (обычно это каталог, в который Вы попадаете сразу после входа в систему). Если каталог задан полным маршрутным именем, он становится текущим. По отношению к новому каталогу нужно иметь право на выполнение, которое в данном случае трактуется как разрешение на поиск.

Поиск образца в файле (grep)

```
grep pattern [file...]
```

Поиск участков текста в файле(ах), соответствующих шаблону `pattern`, где `pattern` может быть как обычной строкой, так и регулярным выражением.

Если файл(ы) для поиска не задан, то команда `grep` работает как фильтр для устройства `stdout`, например в pipes:

```
bash$ ps ax | grep clock
765 tty1     S      0:00 xclock
901 pts/1    S      0:00 grep clock
```

Выдача информации о файлах или каталогах (ls)

```
ls [флаги] [имя ...]
```

Команда ls для каждого имени каталога распечатывает список входящих в этот каталог файлов; для файлов – повторяется имя файла и выводится дополнительная информация в соответствии с указанными флагами. По умолчанию имена файлов выводятся в алфавитном порядке. Если имена не заданы, выдается содержимое текущего каталога. Если заданы несколько аргументов, то они сортируются по алфавиту, однако сначала всегда идут файлы, а потом каталоги с их содержимым.

Изменение режима доступа к файлам (chmod)

`chmod режим файл`

Права доступа к указанным файлам (среди которых могут быть каталоги) изменяются в соответствии с указанным режимом. Режим может быть задан в абсолютном или символьном виде.

Использование символьного вида основано на однобуквенных обозначениях, которые определяют класс доступа и права доступа для членов данного класса. Права доступа к файлу зависят от идентификатора пользователя и идентификатора группы, в которую он входит. Режим в целом описывается в терминах трех последовательностей, по три буквы в каждой:

Владелец	Группа	Прочие
(u)	(g)	(o)
rwx	rwx	rwx

Здесь владелец, члены группы и все прочие пользователи обладают правами чтения файла, записи в него и его выполнения.

Для задания режима доступа в символьном виде используется следующий синтаксис:

`[<кому>] <операция> <права>`

Часть [<кому>] есть комбинация букв u, g и o (владелец, члены группы и прочие пользователи соответственно). Если часть кому опущена или указано a, то это эквивалентно ugo.

Операция может быть: + (добавить право), – (лишить права), = (в пределах данного класса присвоить права абсолютно, то есть добавить указанные права и отнять неуказанные).

Права – любая осмысленная комбинация следующих букв:

r право на чтение;

w право на запись;

x право на выполнение (поиск в каталоге);

s при выполнении переустанавливать действующий идентификатор пользователя или группы;

t после выполнения программы сохранять сегмент команд (бит навязчивости);

l учет блокировки доступа;

Опустить часть <права> можно только для операции «=» для лишения всех прав.

Если надо сделать более одного указания об изменении прав, то при использовании символьного вида в правах не должно быть пробелов, а указания должны разделяться запятыми. Например, команда `chmod u+w,go+x f1` добавит для владельца право писать в файл f1, а для членов группы и прочих пользователей – право выполнять файл. Права устанавливаются в указанном порядке. Право s можно добавлять только для пользователя и группы, право t – только для пользователя.

Чтобы установить права, позволяющие владельцу читать и писать в файл, а членам группы и прочим пользователям только читать, надо использовать следующую запись:

```
chmod u=rw,go=r f1
```

Позволить всем выполнять файл f2

```
chmod +x f2
```

Копирование файлов (cp)

```
cp <файл1> [<файл2> ...] <целевой_файл>
```

Команда cp копирует <файл1> в <целевой_файл>. Файл-источник не должен совпадать с целевым_файлом (будьте внимательны при использовании метасимволов shell'a). Если <целевой_файл> является каталогом, то <файл1>, <файл2>, ..., копируются в него под своими именами. Только в этом случае можно указывать несколько исходных файлов.

Если <целевой_файл> существует и не является каталогом, его старое содержимое теряется. Режим, владелец и группа целевого_файла при этом не меняются.

Если <целевой_файл> не существует или является каталогом, новые файлы создаются с теми же режимами, что и исходные (кроме бита навязчивости, если Вы не суперпользователь). Время последней модификации целевого_файла (и последнего доступа, если он не существовал), а также время последнего доступа к исходным файлам устанавливается равным времени, когда выполняется копирование. Если целевой_файл был ссылкой на другой файл, все ссылки сохраняются, а содержимое файла изменяется.

Перемещение (переименование) файлов (mv)

```
mv [-f] <файл1> [<файл2> ...] <целевой_файл>
```

Команда mv перемещает (переименовывает) <файл1> в <целевой_файл>. <Файл1> не должен совпадать с <целевым_файлом>. Если <целевой_файл> является каталогом, то <файл1>, <файл2>, ..., перемещаются в него под своими именами. Только в этом случае можно указывать несколько исходных файлов.

Если <целевой_файл> существует и не является каталогом, его старое содержимое теряется. Если при этом обнаруживается, что в <целевой_файл> не разрешена запись, то выводится режим этого файла (см. chmod) и запрашивается строка со стандартного ввода. Если эта строка начинается с символа `y`, то требуемые действия все же выполняются, при условии, что у пользователя достаточно прав для удаления <целевого_файла>. Если была указана опция `-f` или стандартный ввод назначен не на терминал, то требуемые действия выполняются без всяких запросов. Вместе с содержимым <целевой_файл> наследует режим

<файла1>.

Если <файл1> является каталогом, то он переименовывается в <целевой_файл>, только если у этих двух каталогов общий надкаталог; при этом все файлы, находившиеся в <файле1>, перемещаются под своими именами в <целевой_файл>. Если <файл1> является файлом, а <целевой_файл> – ссылкой, причем не единственной, на другой файл, то все остальные ссылки сохраняются, а <целевой_файл> становится новым независимым файлом.

Удаление файлов (rm)

```
rm [-f] [-i] <файл> ...
rm -r [-f] [-i] <каталог> ... [<файл> ...]
```

Команда rm служит для удаления указанных имен файлов из каталога. Если заданное имя было последней ссылкой на файл, то файл уничтожается. Для удаления пользователь должен обладать правом записи в каталог; иметь право на чтение или запись файла не обязательно. Следует заметить, что при удалении файла в Linux, он удаляется навсегда. Здесь нет возможностей вроде "мусорной корзины" в windows 95/98/NT или команды undelete в DOS. Так что, если файл удален, то он удален!

Если нет права на запись в файл и стандартный ввод назначен на терминал, то выдается (в восьмеричном виде) режим доступа к файлу и запрашивается подтверждение; если оно начинается с буквы y, то файл удаляется, иначе – нет. Если стандартный ввод назначен не на терминал, команда rm ведет себя так же, как при наличии опции `-f`.

Допускаются следующие три опции:

`-f` Команда не выдает сообщений, когда удаляемый файл не существует, не запрашивает подтверждения при удалении файлов, на запись в которые нет прав. Если нет права и на запись в каталог, файлы не удаляются. Сообщение об ошибке выдается лишь при попытке удалить каталог, на запись в который нет прав (см. опцию -r);

`-r` Происходит рекурсивное удаление всех каталогов и подкаталогов, перечисленных в списке аргументов. Сначала каталоги опустошаются, затем удаляются. Подтверждение при удалении файлов, на запись в которые нет прав, не запрашивается, если задана опция -f или стандартный ввод не назначен на терминал и не задана опция -i. при помощи команда rm можно удалять непустые каталоги, в отличие от команды rmdir, которая удаляет только пустой каталог;

`-i` Перед удалением каждого файла запрашивается подтверждение. Опция -i устраняет действие опции -f; она действует даже тогда, когда стандартный ввод не назначен на терминал.

Опция -i часто используется совместно с -r. По команде:

```
rm  -ir catalog1
```

запрашивается подтверждение:

```
directory catalog1: ?
```

При положительном ответе запрашиваются подтверждения на удаление всех каждого файла в каталоге (включая и подкаталоги), а затем подтверждение на удаление самого каталога.

Удаление каталогов (rmdir)

```
rmdir [-p] [-s] каталог ...
```

Команда rmdir удаляет указанные каталоги, которые должны быть пустыми. Для удаления каталога вместе с содержимым следует воспользоваться командой rm с опцией -r. Текущий каталог [см. pwd] не должен принадлежать поддереву иерархии файлов с корнем – удаляемым каталогом.

Для удаления каталогов нужно иметь те же права доступа, что и в случае удаления обычных файлов [см. rm].

Командой rmdir обрабатываются следующие опции:

`-p` позволяет удалить каталог и вышележащие каталоги, оказавшиеся пустыми. На стандартный вывод выдается сообщение об удалении всех указанных в маршруте каталогов или о сохранении части из них по каким-либо причинам;

`-s` подавление сообщения, выдаваемого при действии опции -p.

Создание ссылки на файл (ln)

Синтаксис команды:

```
ln [-f] <файл1> [<файл2> ...] <целевой_файл>
```

Команда ln делает целевой_файл ссылкой на файл1. <Файл1> не должен совпадать с <целевым_файлом>. Если <целевой_файл> является каталогом, то в нем создаются ссылки на <файл1>, <файл2>, ... с теми же именами. Только в этом случае можно указывать несколько исходных файлов.

Если <целевой_файл> существует и не является каталогом, его старое содержимое теряется. Если при этом обнаруживается, что в <целевой_файл> не разрешена запись, то выводится режим доступа к этому файлу [см. chmod] и запрашивается строка со стандартного ввода. Если эта строка начинается с символа y, то требуемые действия все же выполняются, при условии что у пользователя достаточно прав для удаления <целевого_файла>. Если была указана опция -f или стандартный ввод назначен не на терминал, то требуемые действия выполняются без всяких запросов. <Целевой_файл> наследует режим доступа к файлу1.

Команда ln не создает ссылок между разными файловыми системами, поскольку они (файловые системы) могут добавляться и удаляться.

Создание каталога (mkdir)

```
mkdir [-m режим_доступа] [-p] каталог ...
```

По команде mkdir создается дин или несколько каталогов с режимом доступа 0777 [возможно измененном с учетом umask и опции -m]. Стандартные файлы (. – для самого каталога и .. – для вышележащего) создаются автоматически; их нельзя создать по имени. Для создания каталога необходимо располагать правом записи в вышележащий каталог.

Идентификаторы владельца и группы новых каталогов устанавливаются соответственно равными реальным идентификаторам владельца и группы процесса.

Опции:

`-m` режим_доступа – явное задание режима_доступа для создаваемых каталогов [см. chmod];

`-p` – перед созданием нового каталога предварительно создаются все несуществующие вышележащие каталоги.

Вывод аргументов в стандартный поток вывода(echo)

`echo [опции] [`*`string`* `...]`

Опции:

`-n` – не выводить завершающий символ новой строки.

`-e` – разрешить интерпретацию следующих backslashescaped последовательностей в строках:

`\a` alert (звонок)
`\b` backspace
`\c` запретить завершающий символ новой строки
`\f` перегон страницы
`\n` новая строка
`\r` перевод строки
`\t` горизонтальная табуляция
`\v` вертикальная табуляция
`\\` обратный слэш

Команда echo предназначена для выдачи на стандартный вывод строки символов, которая задана ей в качестве аргумента.

Передаваемая строка может быть перенеправлена в файл с использованием оператора перенаправления вывода «>» . Например:

```
$echo "Hello, world!" > myfile
```

1.3 Экспериментальная часть

1.3.1 Задание на работу

1. Выполнить следующие команды с различными опциями: who, top, ps, df, free, uptime, date.

2. В своем домашнем каталоге создать структуру файлов.

1.3.2 Методические указания по выполнению работы

Запустить Konsole. Эта программа предназначена для выполнения функций командной строки ОС Linux. Здесь в интерактивном режиме вы можете выполнять любые команды и программы, зарегистрированные в системе.

Выполнить команды who, top, ps, df, free, uptime, date (назначение команд и их опции описаны во введении к лабораторной работе).

Войти в свой домашний каталог. Для этого нужно выполнить команду

```
cd ~
```

Здесь хранятся ваши пользовательские файлы и настройки программ, которые вы используете.

Создать следующую структуру каталогов и файлов

1. В домашнем каталоге создать каталог **inform**
2. Перейти в каталог и **inform** создать в нем каталог **lab1**
3. Внутри каталога **lab1** создать каталог **catalog1**, файл **file1(например, используя команду echo)**, каталог **catalog2. Перейти в каталог catalog2.**
4. Внутри каталога **catalog2** создать файлы **file3 и file4** , каталог **catalog3**
5. Внутри каталога **catalog3** создать файл **file5,** попробовать создать жесткую ссылку на файл **file1,** жесткую ссылку на каталог **catalog2.**
6. Создать в каталоге **lab1** символичесткую ссылку **s_link** на файл **file5**

Изменить права доступа к файлу **file3.**

Запустить программу MC (Midnight Commander). Для этого в командной строке необходимо выполнить следующую команду:

```
mc
```

Здесь вы можете посмотреть структуру созданных вами каталогов и просмотреть содержимое файлов.

1.3.3 Содержание отчета

Результаты работы оформить в виде отчета, который должен содержать:

- титульный лист (приложение 1);
- цель работы;
- задание на лабораторную работу;
- описание используемых программ;
- вид командной строки на каждую программу и результат выполнения программ;
- заключение.

Лабораторная работа №2. Создание скрипта в shell

2.1 Цель работы

Получение практических навыков программирования в оболочке Linux, изучение базовых операторов языка программирования в shell. Создание диалогового приложения.

2.2 Теоретическая часть

Скрипт оболочки – это файл, содержащий команды оболочки. Скрипты можно выполнять как обычные команды. Если при запуске такого файла заданы аргументы, на время выполнения скрипта они становятся позиционными параметрами.

В каждом файле, задающем скрипт первая строка имеет вид:

```
#!/bin/bash
```

Это означает, что когда мы запускаем скрипт на выполнение как обычную команду, `/bin/bash` будет выполнять ее для нас.

Помимо использования позиционных параметров, возможно использование и других переменных, определяемых внутри скрипта.

Например:

`fruit = apple` (определение);

`echo $fruit` (доступ);

Ввод с клавиатуры

Переменные можно считывать со стандартного ввода. Для этого используется команда `read`

```
echo -n Enter number of elements:
read x
```

Управляющие структуры shell

Оболочка bash поддерживает операторы выбора `if … then … else` и `case`, а также операторы организации циклов `for, while, until`, благодаря чему она превращается в мощный язык программирования.

Операторы if и test (или [ ])

`if list1 then list2 [ elif list then list ] ... [ else list ] fi`

Оператор `if` проверяет значение, возвращаемое командами из `list1`. Если в этом списке несколько команд, то проверяется значение,

возвращаемое последней командой списка. Если это значение равно 0, то будут выполняться команды из `list2`; если это значение не нулевое,в противном случае будут проверяться последующие условия, и если статус выхода нулевой, то выполнятся соответствующие операторы. Значение, возвращаемой таким составным оператором `if`, совпадает со значением, выдаваемым последней командой выполняемой последовательности.

В качестве выражения, которое стоит сразу после `if` или `elif`, часто используется команда test, которая может обозначаться также квадратными скобками [ ]. Команда `test` выполняет вычисление некоторого выражения и возвращает значение 0, если выражение истинно, и 1 в противном случае. Выражение передается программе `test` как аргумент. Вместо того, чтобы писать

`test expression`,

можно заключить выражение в квадратные скобки:

`[ expression ]`.

`test` и `[` — это два имени одной и той же программы (только синтаксис [требует, чтобы была поставлена закрывающая скобка). Вместо `test` в конструкции `if` может быть использована любая программа.

Пример использования оператора if:

```
if [ -e textmode2.htm ] ; then
    ls textmode*
    else pwd
fi
```

Об операторе test (или […]) надо бы поговорить особо.

Оператор test и условные выражения

Условные выражения, используемые в операторе `test`**,** строятся на основе проверки файловых атрибутов, сравнения строк и обычных арифметических сравнений числовых значений. Сложные выражения строятся из следующих унарных или бинарных операций (для каждого типа проверок существуют свои примитивы):

Проверка файловых атрибутов

- `-a file` Верно, если файл с именем `file` существует.
- `-b file` Верно, если file существует и является специальным файлом блочного устройства.
- `-c file` Верно, если file существует и является специальным файлом символьного устройства.
- `-d file` Верно, если file существует и является каталогом.
1. `-e file` Верно, если файл с именем file существует.
- `-f file` Верно, если файл с именем file существует и является обычным файлом.

- `-g file` Верно, если файл с именем file существует и для него установлен бит смены группы.

3. `-h file` или `-L file` Верно, если файл с именем file существует и является символической ссылкой.

1) `-k file` Верно, если файл с именем file существует и для него установлен "sticky" bit.

- `-p file` Верно, если файл с именем file существует и является именованным каналом (FIFO).
- `-r file` Верно, если файл с именем file существует и для него установлено право на чтение
- `-s file` Верно, если файл с именем file существует и его размер больше нуля.

1. `-t fd` Верно, если дескриптор файла fd открыт и указывает на терминал.

- `-u file` Верно, если файл с именем file существует и для него установлен бит смены пользователя.
- `-w file` Верно, если файл с именем file существует и для него установлено право на запись.

1. `-x file` Верно, если файл с именем file существует и является исполняемым.

- `-O file` Верно, если файл с именем file существует и его владельцем является пользователь, на которого указывает эффективный идентификатор пользователя.

- `-G file` Верно, если файл с именем file существует и принадлежит группе, определяемой эффективным идентификатором группы.

1. `-S file` Верно, если файл с именем file существует и является сокетом.

- `-N file` Верно, если файл с именем file существует и изменялся с тех пор, как был последний раз прочитан.
- `file1 -nt file2` Верно, если файл file1 имеет более позднее время модификации, чем file2.

1) `file1 -ot file2` Верно, если файл file1 старше, чем file2.

- `file1 -ef file2` Верно, если файлы file1 и file2 имеют одинаковые номера устройств и индексных дескрипторов (inode).

Оценка строк

- `-z string` Верно, если длина строки равна нулю.
- `-n string` Верно, если длина строки не равна нулю.
- `string1 == string2` Верно, если строки совпадают. Вместо == может использоваться =.

- `string1 !== string2` Верно, если строки не совпадают.

- `string1 < string2` Верно, если строка string1 лексикографически предшествует строке string2 (для текущей локали).

- `string1 > string2` Верно, если строка string1 лексикографически стоит после строки string2 (для текущей локали).

Арифметические сравнения

- `arg1 OP arg2` Здесь OP — это одна из операций арифметического сравнения: `-eq` (равно), `-ne` (не равно), `-lt` (меньше чем), `-le` (меньше или равно), `-gt` (больше), `-ge` (больше или равно). В качестве аргументов могут использоваться положительные или отрицательные целые.

Из этих элементарных условных выражений можно строить сколь угодно сложные с помощью обычных логических операций ОТРИЦАНИЯ, И и ИЛИ:

- `!(expression)`
  Булевский оператор отрицания.
- `expression1 -a expression2`
  Булевский оператор AND (И). Верен, если верны оба выражения.

1. `expression1 -o expression2`
   Булевский оператор OR (ИЛИ). Верен, если верно любое из двух выражений.

Такие же условные выражения используются и в операторах while и until.

Оператор case

Формат оператора case таков:

```
case word in [ [(] pattern [ | pattern ] ... ) list ;; ] ... esac
```

Команда case вначале производит раскрытие слова word, и пытается сопоставить результат с каждым из образцов pattern поочередно. После нахождения первого совпадения дальнейшие проверки не производятся, выполняется список команд, стоящий после того образца, с которым обнаружено совпадение. Значение, возвращаемое оператором, равно 0, если совпадений с образцами не обнаружено. В противном случае возвращается значение, выдаваемое последней командой из соответствующего списка.

Каждая строка с условием должна завершаться правой (закрывающей) круглой скобкой «)».Каждый блок команд, обрабатывающих заданное условие, должен завершаться *двумя* символами точка-с-запятой «;;».

```
#!/bin/bash
echo -n " Какую оценку ты получил на экзамене?: "
```

```
read z
case $z in
    5) echo Отлично!!!!
    ;;
    4) echo Хорошо !
    ;;
    3) echo Удовлетворительно!
    ;;
    2) echo Неудовлетворительно!
    ;;
    *) echo !
    ;;
 esac
```

Оператор select

Оператор select позволяет организовать интерактивное взаимодействие с пользователем. Он имеет следующий формат:

```
select name [ in word; ] do list ; done
```

Вначале из шаблона word формируется список слов, соответствующих шаблону. Этот набор слов выводится в стандартный поток ошибок, причем каждое слово сопровождается порядковым номером. Если шаблон word пропущен, таким же образом выводятся позиционные параметры. После этого выдается стандартное приглашение bash, и оболочка ожидает ввода строки на стандартном вводе. Если введенная строка содержит число, соответствующее одному из отображенных слов, то переменной name присваивается значение, равное этому слову. Если введена пустая строка, то номера и соответствующие слова выводятся заново. Если введено любое другое значение, переменной name присваивается нулевое значение. Введенная пользователем строка запоминается в переменой REPLY. Список команд list выполняется с выбранным значением переменной name.

```
#!/bin/bash
echo "Какой ОС Вы пользуетесь?"
select var in "Linux" "MS Windows" "Free BSD"
"Other"; do
break
done
echo "Вы бы выбрали $var"
```

Если сохранить этот текст в файле, сделать файл исполняемым и запустить, на экран будет выдан следующий запрос:

Какой ОС Вы пользуетесь?

1) Linux

2) MS Windows

3) Free BSD
4) Other
#?
Если вы введете 1, то появится собщение:
“Вы бы выбрали Linux”

Оператор for

Оператор for работает немного не так, как в обычных языках программирования. Вместо того, чтобы организовывать увеличение или уменьшение на единицу значения некоторой переменной при каждом проходе цикла, он при каждом проходе цикла присваивает переменной очередное значение из заданного списка слов. В целом конструкция выглядит примерно так:

```
for name in words do list done.
```

Правила построения списков команд (list) такие же, как и в операторе if.

Пример. Следующий скрипт создает файлы `file_1, file_2 и file_3`:

```
for a in 1 2 3 ; do
touch file_$a
done
```

В общем случае оператор for имеет формат:

```
for name [ in word; ] do list ; done
```

Вначале производится раскрытие слова word в соответствии с правилами раскрытия выражений, приведенными выше. Затем переменной name поочередно присваиваются полученные значения, и каждый раз выполняется список команд list. Если "in word" пропущено, то список команд list выполняется один раз для каждого позиционного параметра, который задан.

В Linux имеется программа seq, которая воспринимает в качестве аргументов два числа и выдает последовательность всех чисел, расположенных между заданными. С помощью этой команды можно заставить for в bash работать точно так же, как аналогичный оператор работает в обычных языках программирования. Для этого достаточно записать цикл for следующим образом:

```
for a in $( seq 1 10 ) ; do
cat file_$a
done
```

Эта команда выводит на экран содержимое 10-ти файлов: "file_1", ..., "file_10".

Операторы while и until

Оператор while работает подобно if, только выполнение операторов

из списка list2 циклически продолжается до тех пор, пока верно условие, и прерывается, если условие не верно. Конструкция выглядит следующим образом:

```
while list1 do list2 done
```

Пример:

```
while [ -d mydirectory ] ; do
ls -l mydirectory >> logfile
echo /////////////////////// >> logfile
sleep 60
done
```

Такая программа будет протоколировать содержание каталога "mydirectory" ежеминутно до тех пор, пока директория существует.

Оператор until аналогичен оператору while:

```
until list1 do list2 done.
```

Отличие заключается в том, что результат, возвращаемый при выполнении списка операторов list1, берется с отрицанием: list2 выполняется в том случае, если последняя команда в списке list1 возвращает ненулевой статус выхода.

2.2 Экспериментальная часть

2.2.1 Задание на лабораторную работу

Создать скрипт, осуществляющий вывод меню, состоящего из следующих пунктов:

– текущий пользователь;
– объем используемой памяти;
– объем дискового пространства;
– запущенные процессы;
– процессы, запущенные текущим пользователем;
– системная дата и время;
– время запуска системы;
– выход;
– вывод соответствующей информации в зависимости от выбранного пункта меню. Процесс повторять до тех пор, пока не будет выбран пункт меню «выход».

2.2.2 Методические указания по выполнению работы

В папке inform создать файл lab2. Сделать этот файл выполняемым.

В начало файла поместить следующую строку:

```
#!/bin/bash
```

Для обеспечения диалога с пользователем использовать оператор select, где вывести все пункты меню.

Сделать обработку всех пунктов меню, воспользовавшись оператором case или оператором if и командами оболочки.

Выход из программы выполнить при помощи оператора break.

В ходе выполнения работы обратите внимание на следующие пункты:

1) для названий пунктов меню лучше использовать латинский алфавит. В названиях обязательно обращать внимание на специальные символы и соответствующие правила их использования в строках;

2) необходимо обращать внимание на регистр вводимых символов (в командах и их опциях, в специальных словах языка shell), на знаки пробела. Например, в операторе `[ ]`:

```
[ ␣ expression ␣ ];
```

3) для работы с меню удобно использовать операторы `select`, `case`, `if` (описание этих операторов приведены в документе с лекциями).

2.2.3 Содержание отчета

Результаты работы оформить в виде отчета, который должен содержать:

- титульный лист (приложение А);
- цель работы;
- задание на лабораторную работу;
- описание используемых команд и операторов;
- листинг скрипта и результаты выполнения;
- заключение.

Раздел 2. Scilab

Scilab – это свободно распространяемая система компьютерной математики, которая предназначена для выполнения инженерных и научных вычислений, таких как решение нелинейных уравнений и систем, решение задач линейной алгебры, решение задач оптимизации, дифференцирование и интегрирование, решение обыкновенных дифференциальных уравнений и систем, задачи обработки экспериментальных данных (интерполяция и аппроксимация, метод наименьших квадратов).

Scilab предоставляет широкие возможности по созданию и редактированию различных видов графиков и поверхностей.

Система имеет достаточно мощный собственный язык программирования высокого уровня.

Scilab был разработан в 1994 году во Франции, в Национальном исследовательском институте информатики и автоматизации (Institut national de recherche en informatique et en automatique, INRIA) и Национальной школе дорожного ведомства (École Nationale des Ponts et Chaussées, ENPC). С 2003 года поддержкой Scilab занимается консорциум Scilab Consortium.

Отличительные особенности пакета:

- бесплатность;
- маленький размер (13Мб против более чем двухгигабайтного пакета MATHLAB);
- возможность запуска в консоли без использования графического интерфейса. Это позволяет производить автоматизированные вычисления.

В рамках курса информатики струдентам предлагается изучение основных принципов работы с пакетом, связанных с вычислением математических выражений, построением графиков функций, работой в векторами и матрицами, а также основ программирования в Scilab.

Лабораторная работа №3. Основы работы в Scilab

3.1 Цель работы

Знакомство с основными принципами работы пакета Scilab: расчет значений выражений, построение графиков Scilab и знакомство с некоторыми операторами языка программирования в Scilab.

3.2 Теоретическая часть

Элементарные математические выражения. Использование встроенных переменных и функций

Для выполнения *простейших арифметических операций* в Scilab применяют следующие операторы:+ сложение, – вычитание, * умножение, / деление слева направо, \ деление справа налево, ^ возведение в степень.

Вычислить значение арифметического выражения можно, если ввести его в командную строку и нажать клавишу `ENTER`. В рабочей области появится результат.

Если символ точки с запятой «;» указан в конце выражения, то результат вычислений не выводится, а активизируется следующая командная строка.

В рабочей области **Scilab** можно определять *переменные*, а затем использовать их в выражениях.

Любая переменная до использования в формулах и выражениях должна быть определена. Для определения переменной необходимо набрать имя переменной, символ «=» и значение переменной. Здесь знак равенства – это *оператор присваивания*, действие которого не отличается от аналогичных операторов языков программирования. То есть, если в общем виде оператор присваивания записать как

```
имя переменной = значение выражения
```

то в *переменную*, имя которой указано слева, будет записано *значение выражения*, указанного справа.

Scilab содержит ноторое количество уже определенных переменных. Все *системные переменные* в Scilab начинаются с символа `%`: мнимая единица; число π; число e=2.7182818 и др.

Пакет Scilab снабжен достаточным количеством всевозможных *встроенных функций*.

Функция	Описание функции
Тригонометрические	
sin(x)	синус числа x
cos(x)	косинус числа x
tan(x)	тангенс числа x
cotg(x)	котангенс числа x
asin(x)	арксинус числа x
acos(x)	арккосинус числа x
atan(x)	арктангенс числа x
Экспоненциальные	
exp(x)	Экспонента числа x
log(x)	Натуральный логарифм числа x
Другие	
sqrt(x)	корень квадратный из числа x

abs(x)	модуль числа x
log10(x)	десятичный логарифм от числа x
log2(x)	логарифм по основанию два от числа x

Построение графиков

Функция plot2d

В общем виде обращение к функции имеет вид:

```
plot2d([loglog],x,y,[key1=value1,key2=value2,
..., keyn=valuen]
```

- `logflag` – строка из двух символов, каждый из которых определяет тип осей (n – нормальная ось, l – логарифмическая ось), по умолчанию "nn";
- `x` – массив абсцисс;
- `y` – массив ординат(или матрица, каждый столбец которого содержит массив ординат очередного графика) (количество элементов в массиве x и y должно быть одинаковым), если x и y – являются матрицами одного размера, то в этом случае, каждый столбец матрицы y отображается относительно соответствующего столбца матрицы **x**;
- `keyi=valuei` – последовательность значений параметров графиков, возможны следующие значения параметров: `style` – определяет массив (`mas`) числовых значений цветов графика (`id` цвета), количество элементов массива совпадает с количеством изображаемых графиков, по умолчанию, по умолчанию представляет собой массив mas_i=i, цвет i-й линии совпадает с номером i, для формирования id соответствующего цвета (`кода цвета`) можно воспользоваться функцией `color`, которая по названию (`color("цвет")`) или коду `grb` (`color(r,g,b)`) цвета формирует нужный `id` (код) цвета. Если значение стиля отрицательное то это будет точечный график без соединения точек между собой линиями. Пример построения нескольких графиков различного цвета приведен ниже

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"), color("blue")]);
```

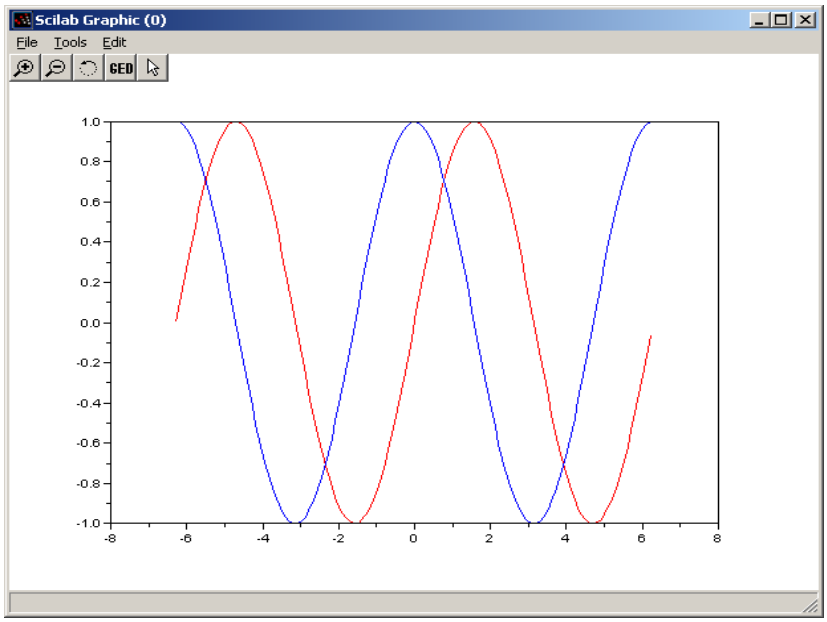


Рисунок 3.1 –Использование параметра style в функции plot2d

- `rect` – этот вектор `[xmin, ymin, xmax, ymax]` определяет размер окна вокруг графика.

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),color("blue")],
rect=[-8,-2,8,2]);
```

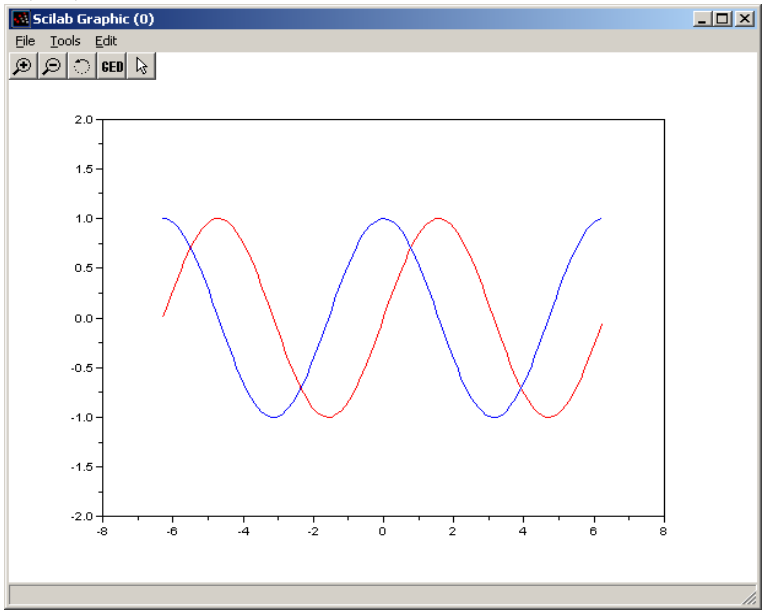


Рисунок 3.2 –Использование параметра rect

- `frameflag` – параметр определяет окно в котором, будет изображаться график, он может принимать следующие значения: 0 – не вычислять размеры окна, использовать значения по умолчанию или значения из предыдущего графика, 1 – размер окна определяется параметром `rect`, 2 – размер окна определяется из соотношения между

минимальным или максимальным значениями `x` и `y`, 3 – размер окна определяется параметром `rect` в в изометрическом масштабе, 4– размер окна определяется из соотношения между минимальным или максимальным значениями `x` и `y` в изометрическом масштабе,

- `axesflag` – параметр, который определяет рамку вокруг графика, следует выделить следующие значения этого параметра: 0 – нет рамки вокруг графика (1 – изображение рамки, ось **y** слева; 3 – изображение рамки, ось **y** справа; 5 – изображение осей проходящих через точку (0,0) (см. следующие рисунки);

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),  color("blue")], axesflag = 0);
```

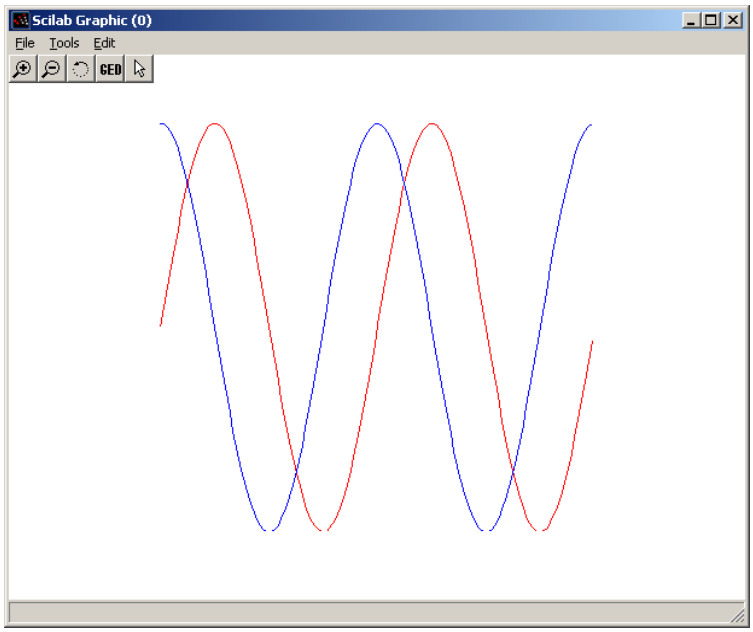


Рисунок 3.3 – Параметр axesflag=0 в функции plot2d

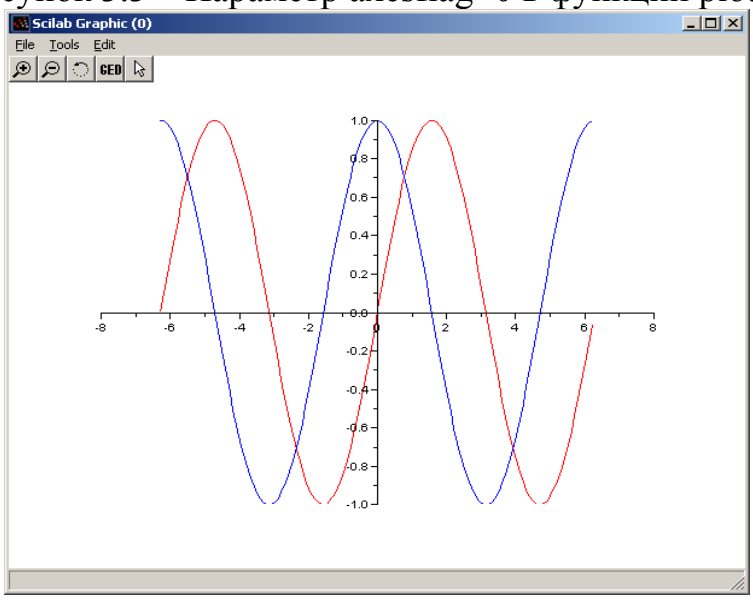


Рисунок 3.4 – Параметр axesflag=5 в функции plot2d

- nax – этот параметр используют, если параметр axesflag равен 1, nax представляет массив из четырех значений [nx, Nx, ny, Ny] – где Nx (Ny) – число основных делений с подписями под осью X (Y), nx (ny) – число промежуточных делений;
- leg – строка, определяющая легенды для каждого графика, структура строки такая: "leg1@leg2@leg3@...@legn", где leg1 – легенда первого графика, ..., legn – легенда первого графика.

Пример построения графиков функций с использованием параметра nax при построении функции plot2d.

```
x=[-8:0.1:8];
y=[sin(x); cos(x)];
```

Пример построения графиков функции с использованием параметров nax и leg:

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"), color("blue")],
axesflag=5, nax=[4,9,3,6],leg="sin(x)@cos(x)");
```

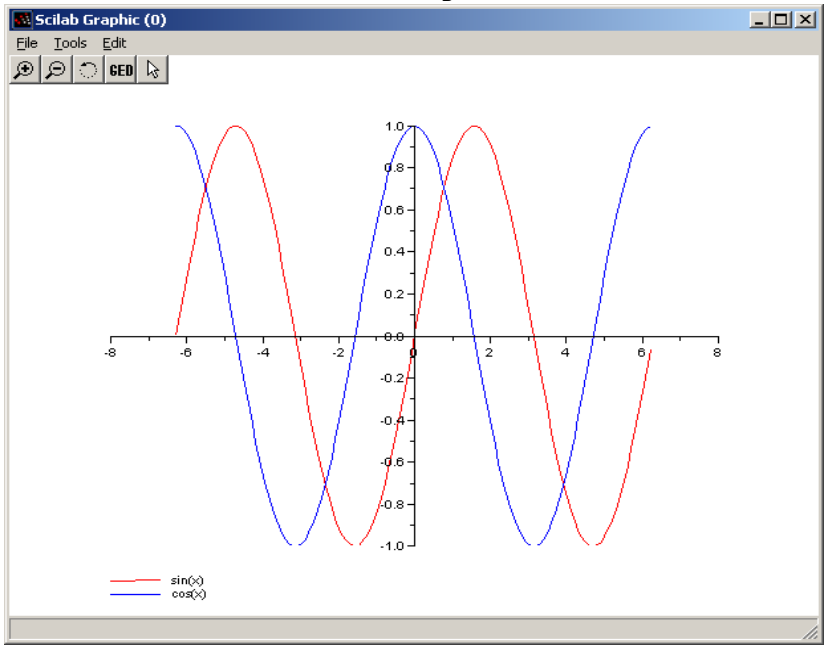


Рисунок 3.5 – Использование параметров leg и nax в функции plot2d

Функцию plot2d можно использовать для построения точечных графиков. В этом случае обращение к функции имеет вид:

```
plot2d(x,y,d),
```

где d – отрицательное число, определяющее тип маркера

```
x=[-2*%pi:0.25:2*%pi];
y=sin(x);
plot2d(x,y,-3);
```

Изображение сетки на графике

Для изображения сетки следует воспользоваться функцией `xgrid(color)`, где `color` определяет `id` цвета линии сетки.

Построение полярных графиков

Для построения графиков в полярной системе координат в Scilab служит функция polarplot

`polarplot(fi,ro,[key1=value1,key2=value2, ..., keyn=valuen])`

де fi – полярный угол, ro – полярный радиус.

Рассмотрим пример построения полярных графиков $\rho = 3\cos(5\varphi), \rho_1 = 3\cos(3\varphi)$

```
fi=0:0.01:2*%pi;
ro=3*cos(5*fi);
ro1=3*cos(3*fi);
polarplot(fi,ro,style=color("red"));
polarplot(fi,ro1,style=color("blue"));
```

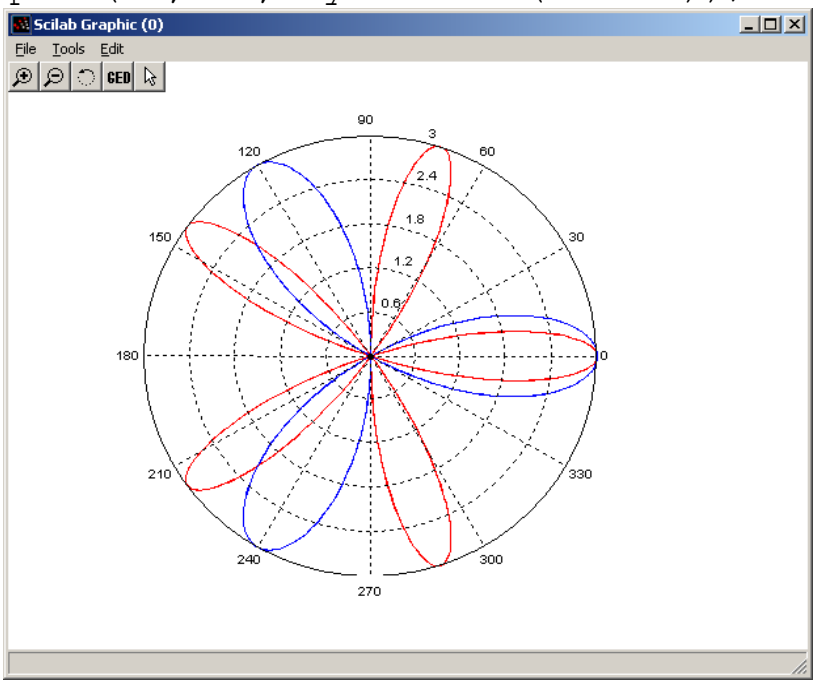
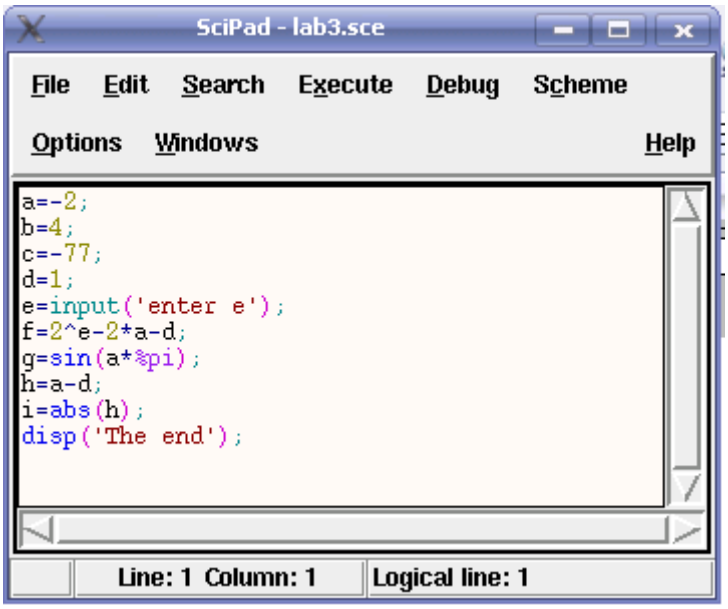


Рисунок 3.6 – Пример построения полярных графиков

Программирование в Scilab

Работа в Scilab может осуществляться в режиме командной строки, но и в так называемом программном режиме. Для создания программы (программу в Scilab иногда называют сценарием) необходимо:

1) вызвать команду **Editor** из меню;
2) в окне редактора **SciPad** набрать текст программы.



3) сохранить текст программы с помощью команды **File-Save** в виде файла с расширением **sce**, например **lab3.sce;**

4) после чего программу можно будет вызвать набрав в командной строке **exec**, например **exec("lab3.sce"),** нажав комбинацию клавиш Ctrl+l или вызвав команду меню **File-Exec...** .

3.2 Экспериментальная часть

3.2.1 Задание на лабораторную работу

1. Вычислить выражения при a = 5, b = 3, c = 0.25:

$$E = 2ab + \frac{\cos c - \sin^2 \frac{\pi * a}{b}}{\sqrt{\left(a^2 + b^{a+c}\right) \cdot c}},$$

$$F := e^{\sin c} \arcsin \frac{b}{a}$$

2. Построить графики функций

1) $y = \cos(2\pi - x)$, $y = \sin \pi x$, $y = \cos^3 \pi x$, $x \in [-4;4]$, шаг 0,1

Исследовать влияние параметров графиков (привести 3-4 примера);

2) построить декартовы и полярные графики следующих функций:

$$X(\alpha) := \cos(a) \cdot \sin(a)$$

$$Y(\alpha) := \sin(a * \pi)^2 - 1, \; a \in [0;2\pi] \text{ шаг } \pi/30.$$

$$P(\alpha) := \cos(a)$$

3. Реализовать программу, осуществляющую вывод решения квадратного уравнения (параметры a,b,c вводятся с клавиатуры).

4. Реализовать программу, осуществляющую вывод значения

следующей функции в зависимости от введенных параметров

$$\mu(x) = \begin{cases} 0, & \text{при } x \le a \\ \dfrac{(x-a)}{(b-a)}, & \text{при } a < x \le b \\ \dfrac{(c-x)}{(c-b)}, & \text{при } b < x \le c \end{cases}$$

3.3.2 Методические указания по выполнению работы

Запустить пакет Scilab.

В основном окне выполнить вычисления выражений из задания 1, воспользовавшись стандартными функциями и константами Scilab.

Построить графики кривых из задания 2, воспользовавшись функциями plot2d и polarplot с соответствующими параметрами, предварительно задав вектора аргумента и значений функций.

Для написания программ воспользоваться программным режимом.

3.2.3 Соджержание отчета

Результаты оформить в виде отчета, который должен содержать

1) описание используемых операторов и функций с соответствующими листингами;

2) скриншоты графиков из задания 2;

3) описание программ и используемых в них операторов языка программирования Scilab; листинги программ.

Лабораторная работа №4. Управляющие структуры и работа с матрицами в Scilab

4.1 Цель работы

Получение практических навыков в программировании на Sci-языке: работа с управляющими структурами, обработка массивов

4.2 Экспериментальная часть

4.2.1 Задание на лабораторную работу

Написать программу, выполняющую по выбору пользователя следующие операции с квадратной матрицей случайных чисел:

- отображение правой половины матрицы на левую зеркально симметрично относительно вертикальной оси;
- отображение матрицы симметрично относительно главной диагонали;
- размещение на главной диагонали сумм элементов, лежащих на той же строке и том же столбце, что и сам диагональный элемент.

После выполнения соответствующей операции необходимо снова вывести приглашение на экран.

4.2.2 Методические указания по выполнению работы

Требования к программе:

– предусмотреть выход из программы.
– обязателен вывод исходной и конечной матрицы.

1. Для организации основного цикла программы можно воспользоваться оператором while:

```
while условие
    операторы
end
```

2. Для организации системы меню можно лучше всего воспользоваться оператором select:

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
…
else операторы
end
```

3. Работу с элементами матрицы необходимо выполнять в цикле, например, используя цикл с параметром

```
for x=xn:hx:xk
операторы
end
```

При работе с массивами в качестве параметра цикла используют индекс массива. . В матрицах используют вложенные циклы

```
for i=in:ik
    for j=jn:jk
    операторы
end
end
```

где `i` и `j`– номер строки и номер столбца соответственно.

4. Для организации выхода из программы можно предусмотреть дополнительный пункт меню «выход».

4.2.3 Содержание отчета

Отчет по лабораторной работе должен содержать

- описание используемых операторов и функций;
- описание программы, листинг программы, скриншоты результатов выполнения программы.

Лабораторная работа №5. Создание пользовательского приложения в Scilab

5.1 Цель работы

Объединение знаний, полученных в результате изучения пакета Scilab. Создание пользовательского приложения, осуществляющего работу с графиками.

5.2 Теоретическая часть

Создание графического окна

Для создания пустого графического окна служит функция figure.

```
F=figure();
```

В результате выполнения этой команды будет создано данном графическое окно с именем objfigure1. По умолчанию первое окно получает имя objfigure1, второе – objfigure2 и т.д. Указатель на графическое окно записывается в переменную F. Размер и положение окна на экране монитора можно задавать с помощью параметра:

'position', [x y dx dy],

где x, y – положение верхнего левого угла окна (по горизонтали и вертикали соответственно) относительно верхнего левого угла экрана;

dx – размер окна по горизонтали (ширина окна) в пикселях;

dy – размер окна по вертикали (высота окна) в пикселях.

Параметры окна можно задавать одним из двух способов.

1. Непосредственно при создании графического окна задаются его параметры. В этом случае обращение к функции `figure` имеет вид

```
F=figure('Свойство1', 'Значение1', 'Свойство2',
 'Значение2', ...,'Свойствоn', 'Значениеn')
```

здесь 'Свойство1' – название первого параметра, Значение1 – его значение, 'Свойство2' – название второго параметра, Значение2 – значение второго параметра и т.д.

Например,

```
F=figure('position', [10 100 300 200]);
```

2. После создания графического окна с помощью функции

```
set(f,'Свойство','Значение')
```

устанавливается значение параметров, здесь `f` – указатель на графическое окно, `'Свойство'` – имя параметра, `'Значение'` – его значение.

```
f=figure();
set(f,'position',[20,40,600,450])
```

Для изменения заголовка окна используется параметр `'figure_name', 'name'` определяющий заголовок окна (`'name'`).

```
f=figure();
```

```
set(f,'position',[20,40,600,450]);
set(f,'figure_name','FIRST WINDOW');

f=figure('position',[20,40,600,450],'figure_name'
,'FIRST WINDOW');
```

Графическое окно можно закрыть с помощью функция `close(f)` (здесь `f` – указатель на окно). Удаляется окно с помощью функции `delete(f)`, где `f`- указатель на окно.

Создание объектов управления (функция uicontrol)

В Scilab используется динамический способ создания интерфейсных компонентов. Он заключается в том, что на стадии выполнения программы могут создаваться (и удаляться) те или иные графические объекты (кнопки, метки, флажки и т.д.) и их свойствам присваиваются соответствующие значения.

```
C=uicontrol(F, 'Style', 'тип_компонента',
'Свойство_1', Значение_1, 'Свойство_2',
Значение_2,... 'Свойство_k',Значение_к);
```

где `C` – указатель на создаваемый компонент;

`F` – указатель на объект, внутри которого будет создаваться компонент; первый аргумент функции `uicontrol` не является обязательным, и если он отсутствует, то родителем (владельцем) создаваемого компонента является текущий графический объект – текущее графическое окно;

`'Style'`- служебная строка `Style`, указывает на стиль создаваемого компонента(символьное имя);

'тип_компонента'- определяет, к какому классу принадлежит создаваемый компонент, это может быть `PushButton`, `Radiobutton`, `Edittext`, `StaticText`, `Slider`, `Panel`, `Button Group`, `Listbox` или др компоненты;

'`Свойство_к`', `Значение_к` – определяют свойства и значения отдельных компонентов, они будут описаны ниже конкретно для каждого компонента.

У существующего интерфейсного объекта можно изменить те или иные свойства с помощью функции `set`:

```
set(C,'Свойство_1',Значение_1,...)
```

где `C` – указатель на компонент, параметры которого будут меняться;

Получить значение параметра компонентов можно с помощью функции `get` следующей структуры:

```
get(C,'Свойство')
```

где `C` – указатель на динамический интерфейсный компонент, значение параметра которого необходимо узнать;

'Свойство' - имя параметра, значение которого нужно узнать.

Функция возвращает значение параметра. Далее мы поговорим об особенностях создания различных компонентов.

Типы компонентов:

`pushbutton` – кнопка;

`text` – текстовое поле для отображения текстовой информации;

`edit` – окно редактирования;

`radiobutton` – кнопка со значением on или off – переключатель;

`checkbox` – флажок;

`listbox` – список строк.

Свойства компонентов:

`string` – заголовок;

`position` – координаты расположения компонента;

`callback` – обработка события (например, при нажатии на кнопку);

`BackgroundColor` – цвет фона (значением является ибо строка, либо вектор, состоящий из трех величин типа real. В случае, если это строка, т значения компонент цвета разделяются с помощью знака «|»: «R|G|B». Каждая величина предствляет значение в интервале [0,1]);

`Horizontalalignment` – выравнивание текста (используется в компонентах `'text'`, `'edit'` и `'checkbox'`, значения: `left` – выравнивание по левому краю; `center` – выравнивание текста по центру (значение по умолчанию); `right` – выравнивание по правому краю).

Пример создания текстового поля.

```
w=figure('Position',[50,50,200,200]);
t=uicontrol('Style','text','Position',[100,100,50
,20],'String', '0.1');
set(t,'BackgroundColor',[1 1 1]);
set(t,'HorizontalAlignment','left');
```

Пример создания кнопки

```
w=figure();
pbtn=uicontrol(w,'Style','pushbutton',   'string',
'OK', 'Callback', 'sinus');
function y=sinus()
x=-5*%pi:0.2*%pi:5*%pi;
y=sin(x);
plot(x,y);
endfunction
```

Пример создания переключателя

```
hFig=figure('Position',[50,50,200,200]);
//Создание радиокнопок
hRb1=uicontrol('Style','radiobutton','String','si
n(x)','value',0,
'Position',[25,100,60,20],'callback','Radio1');
hRb2=uicontrol('Style','radiobutton','String','co
s(x)','value',0,
'Position',[25,140,60,20],'callback','Radio2');

//обработчик нажатия на кнопки
function Radio1()
newaxes;
x=-2*%pi:0.1:2*%pi;
set(hRb2,'value',0);
y=sin(x);
plot(x,y);
xgrid();
endfunction

function Radio2()
newaxes;
x=-2*%pi:0.1:2*%pi;
set(hRb1,'value',0);
y=cos(x);
plot(x,y);
xgrid();
endfunction
```

Пример создания списка строк

```
f=figure();
h=uicontrol(f,'style','listbox','position',[10  10
150 160]);
set(h, 'string', "item 1|item 2|item3");
set(h, 'value', [1 3]);
```

Работа с файлами в Scilab

Открытие файла

```
[fd,err]=mopen(file, mode),
```

где `err` – индикатор ошибки;

`fd` – параметр `fd`, возвращаемый функцией mopen используется как файловый идентификатор

`file` – путь к файлу;

`mode` – режим открытия файла.

Сушествуют следующие режимы:

`r` или `rb`: чтение (файл должен существовать);

`w` или `wb`: запись(если файл не существовал, то он создается, если существовал, то предыдущее содержимое удаляется;

`a` или `ab`: добавление в конец файла (если файл не существовал, то он создается);

`r+` или `r+b`: чтение и запись(файл должен существовать);

`w+` или `w+b`: чтение и запись (принцип работы как в `w` и `wb`);

`a+` или `a+b`: чтение и запись (принцип работы как в `a` и `ab`).

Закрытие файла

`mclose([fd])`

С помощью функции `mclose('all')` можно закрыть сразу все открытые файлы, кроме стандартных системных файлов.

Если идентификатор фаула опущен, то закрывается последний открытый файл.

Запись в файл

Функция `mfprintf`

`mfprintf(fd, format, s),`

где `format` – форматная строка.

`s` – список выводимых параметров.

В форматной строке указываются форматы вывода параметров:

`%[ширина][.точность]тип`.

Тип

`c` При вводе символьный тип char, при выводе один байт.

`d,i` Десятичное со знаком

`i` Десятичное со знаком

`o` Восьмеричное int unsigned

`u` Десятичное без знака

`x, X` Шестнадцатеричное int unsigned, при x используются символы a-f, при X – A-F.

`f` Значение со знаком вида [-]dddd.dddd

`e` Значение со знаком вида [-]d.ddde[+|-]ddd

`E` Значение со знаком вида [-]d.ddddE[+|-]ddd

`g` Значение со знаком типа e или f в зависимости от значения и точности

`G` Значение со знаком типа E или F в зависимости от значения и

точности

`s` Строка символов

Флаги

`-` Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.

`+` Перед числом выводится знак «+» или «-»

Пробел Перед положительным числом выводится пробел, перед отрицательным – «-»

`#` Выводится код системы счисления: 0 – перед восьмеричным числом, 0x (0X) пе-ред шестнадцатеричным числом.

Ширина

`n` Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами.

`0n` То же, что и n, но незаполненные позиции заполняются нулями.

Точность

`ничего` Точность по умолчанию

`n` Для типов e, E, f выводить n знаков после десятичной точки

Модификатор

`h` Для d, i, o, u, x, X короткое целое

`l` Для d, i, o, u, x, X длинное целое

В строке вывода могут использоваться некоторые специальные символы

`\b` Сдвиг текущей позиции влево

`\n` Перевод строки

`\r` Перевод в начало строки, не переходя на новую строку

`\t` Горизонтальная табуляция

`\'` Символ одинарной кавычки

`\''` Символ двойной кавычки

`\?` Символ ?

Функция `mput`

```
mput(x [,type,fd]),
```

где `x`: число или вектор;

`fd`: дескриптор файла;

type: формат записи числа:

"l","i","s","ul","ui","us","d","f","c","uc":
соответственно long, int, short, unsigned long, unsigned int, unsigned short, a double, float, char и unsigned char.

Запись матрицы в файл – функция write

write(filename,a,[format])

a: матрица.

Чтение из файла

Функция mfscanf

A=mfscanf(fd, s),

где из файла с идентификатором fd считываются в переменную A значения в соответствии с форматом s. При чтении числовых значений из текстового файла следует помнить, что два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

Функция mget

x=mget([n,type,fd]),

где x – вектор ли число;

n – число считываемых параметров;

type – формат числа(см mput)

Чтение матрицы – функция read

[x]=read(filename,m,n,[format]),

где m и n – размерности матрицы. m=-1 если заранее неизвестно количество строк матрицы

Функция определения конца файла

e=meof(fd)

Функция определяет, достигнут ли конец файла

5.2 Экспериментальная часть

5.2.1 Задание на лабораторную работу

Создать приложение со следующим функционалом:

- приложение предназначено для построения графиков заранее заданных функций;
- выбор параметров функций производится с помощью диалоговых окон;
- промежуточные результаты расчета функций сохраняются в файле.

Варианты заданий

1) функция 1: f(x)=x sin x, функция 2: f(x)=x2-x+10;
2) функция 1: f(x)=-x3+x2-4, функция 2: f(x)= x sin (π -x);
3) функция 1: f(x)=cos(2x-π), функция 2: f(x)=x4-x2-1;
4) функция 1: f(x)=cos(2x-π/2), функция 2: f(x)=x3-2x+1.

5.2.2 Методические указания по выполнению работы

Создать диалоговое окно со следующими полями ввода:

1) тип функции;
2) начальное значение аргумента функции;
3) конечно е значение аргумента;
4) шаг расчета функции.

Обработать ввод данных пользователя

На основе введенных данных произвести вычисления значений функции.

Записать значения аргумента и соответствуюшие им значения функции в файл с именем output.txt.

Считать значения и соответствующие им значения функции из файла с именем output.txt и построить соответствующий график.

5.2.3 Содержание отчета

Результаты оформить в виде отчета, где привести:

- текст программы на Sci-языке;
- привести содержимое файла output.txt;
- привести график функции;
- повторить для второй функции.

Раздел 3. Язык программирования Pascal

Pascal – язык программирования высокого уровня, разработанный в 1970 г. Николаусом Виртом в качестве языка обучения структурному программированию. В ноябре 1970г вышла первая официальная публикация описания языка с изложением синтаксиса и семантики. Большую роль в развитие Pascal сыграла компания Borland International, создавшая Turbo-среду разработки.

Pascal – один из первых языков, основанный на строгой типизации и принципах структурного программирования.

В качестве компилятора в рамках курса предлагается использовать Free Pascal Compiler. Это свободно распространяемый, кроссплатформенный компилятор языка Pascal с открытыми кодами, совместимый с Turbo Pascal 7.0 и Delphi. В первом семестре языку Pascal посвящена только одна лабораторная работа, изучение языка продолжится во следующем семестре.

Лабораторная работа №6. Применение условных операторов в Pascal

6.1 Цель работы

Получение практических навыков работы с системой FreePascal, изучение структуры программы и базовых операторов языка Pascal. Создание диалогового приложения с использованием операторов ветвления.

6.2 Теоретическая часть

Любая программа на Pascal состоит из трех блоков: блока объявлений, блока описания процедур и функций и блока операторов (основной блок программы)

Блок объявлений:

`program` ... (название программы)
`uses` ... (используемые программой внешние модули)
`const` ... (подраздел описания констант)
`type` ... (подраздел объявления типов)
`var` ... (подраздел объявления переменных)

Блок описания процедур и функций:

```
procedure (function)
begin
    ...
end;
```

...

Блок основной программы:

```
begin
```

(операторы основной программы) ...

```
end;
```

Типы данных

Тип переменной задает вид того значения, которое ей присваивается и правила, по которым операторы языка действуют с переменной. ***Тип константы*** определяется способом записи ее значения.

В Pascal предопределены следующие простейшие типы переменных:

Целочисленные типы

`byte`	целое число от 0 до 255, занимает одну ячейку памяти (байт).
`word`	целое число от 0 до 65535, занимает два байта.
`integer`	целое число от –32768 до 32767, занимает два байта.
`shortint`	целое число от -128 до 127, занимает 1 байт
`longint`	целое число от –2147483648 до 2147483647, занимает четыре байта.

Вещественные типы данных

`real`	число с дробной частью от $2.9*10^{-39}$.до $1.7*10^{38}$, может принимать и отрицательные значения, на экран выводится с точностью до 12-го знака после запятой, если результат какой либо операции с `real` меньше, чем $2.9*10^{-39}$, он трактуется как ноль. Переменная типа `real` занимает шесть байт.
`single`	число с дробной частью от $1.5*10^{-45}$ до $3.4*10^{38}$, может принимать и отрицательные значения, на экран выводится с точностью до 8-го знака после запятой, если результат какой либо операции с `real` меньше, чем $1.5*10^{-45}$, он трактуется как ноль. Переменная типа `real` занимает шесть байт.
`double`	число с дробной частью от $5.0*10^{-324}$.до.$1.7*10^{308}$, может принимать и отрицательные значения, на экран выводится с точностью до 16-го знака после запятой, если результат какой либо операции с `double` меньше, чем $5.0*10^{-324}$, он трактуется как ноль. Переменная типа `double` занимает восемь байт.

Символьный тип

`char`	символ, буква, при отображении на экран выводится тот символ, код которого хранится в выводимой переменной типа `char`, переменная занимает один байт. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII.

Строковый тип

`string`	строка символов, на экран выводится как строка символов, коды которых хранятся в последовательности байт, занимаемой выводимой переменной типа STRING; в памяти занимает от 1 до 256 байт – по количеству символов в строке, плюс один байт, в котором хранится длина самой строки.

Логический тип

`boolean`	логическое значение (байт, заполненный единицами, или нулями), `true`, или `false`.

Арифметические операции и стандартные функции в Pascal

Арифметические операции

Операция	Действие	Тип операндов	Тип результата
бинарные			
+	сложение	целый, вещественный	целый, вещественный
-	вычитание	целый, вещественный	целый, вещественный
*	умножение	целый, вещественный	целый, вещественный
/	деление	целый, вещественный	вещественный
div	целочисленное деление	целый	целый
mod	остаток от деления	целый	целый
унарные			
+	сохранение знака	целый, вещественный	целый, вещественный
-	отрицание знака	целый, вещественный	целый, вещественный

Операции отношения

Операции отношения выполняют сравнение двух операндов и определяют, истинно значение или ложно. Сравниваемые величины могут принадлежать к любому типу данных, и результат всегда имеет логический тип, принимая одно значение из двух: истина или ложь.

Операция	Название	Выражение
=	Равно	A=B
<>	Неравно	A<>B
>	Больше	A>B
<	Меньше	A<B
>=	Больше или равно	A>=B
<=	Меньше или равно	A<=B

Стандартные математические функции

Обращение	Тип аргумента	Тип результата	Функция
abs(x)	целый, вещественный	целый, вещественный	модуль аргумента
arctan(x)	целый, вещественный	вещественный	арктангенс
cos(x)	целый, вещественный	вещественный	косинус
exp(x)	целый, вещественный	вещественный	e^x – экпонента
frac(x)	целый, вещественный	вещественный	дробная часть x
int(x)	целый, вещественный	вещественный	целая часть x
ln(x)	целый, вещественный	вещественный	натуральный логарифм
random		вещественный	псевдослучайное число [0,1]

`random(x)`	целый	целый	псевдослучайное число [0,x]
`round(x)`	вещественный	целый	округление до ближайшего целого
`sin(x)`	целый, вещественный	вещественный	синус
`sqr(x)`	целый, вещественный	вещественный	квадрат x
`sqrt(x)`	целый, вещественный	вещественный	корень квадратный из x
`trunc(x)`	вещественный	целый	ближайшее целое, не превышающее x по модулю

Логические операции

Логические выражения в результате вычисления принимают логические значения True и False. Операндами это выражения могут быть логические константы, переменные, отношения. Идентификатор логического типа в Pascal: boolean.

В Паскале имеется 4 логические операции: отрицание -not, логическое умножение -and, логическое сложение – or, исключающее “или” – xor . Использованы обозначения: T (true), F (false).

A	B	not A	A and B	A or B	A xor B
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T

Приоритеты операций: not, and, or, xor. Операции отношения (= , <>) имеют более высокий приоритет, чем логические операции, поэтому их следует заключать в скобки при использовании по отношению к ним логических операций.

Приоритет операций (в порядке убывания):

- вычисление функции;
- унарный минус, not;
- умножение, деление, div, mod, and;
- сложение, вычитание, or, xor;
- операции отношения

Процедуры ввода/вывода

`write(p1,p2,... pn);` – выводит на экран значения выражений p1,p2,... pn.

Выражения могут быть числовые, строковые, символьные и логические. Под выражением будем понимать совокупность некоторых действий, применённых к переменным, константам или литералам, например: арифметические действия и математические функции для чисел, функции для обработки строк и отдельных символов, логические

выражения и т.п.

Возможен форматный вывод, т. е. явное указание того, сколько выделять позиций на экране для вывода значения.

Например, для того, чтобы вывести значение выражения a+b с выделением для этого 10 позиций, из них 5 – после запятой

```
write(a+b:10:5);
```

Или например, вывести значение выражения p любого другого типа , выделив под него 10 позиций

```
write(p:10);
```

Вывод на экран в любом случае производится по правому краю выделенного поля.

`writeln(p1,p2,... pn);` – аналогично write, выводит значения p1,p2,... pn, после чего переводит курсор на новую строку.

Существует вариант `writeln;` (без параметров), что означает лишь перевод курсора на начало новой строки.

`readln(v1,v2,...vn);` – ввод с клавиатуры значений переменных v1,...vn. Переменные могут иметь строковый, символьный или числовой тип. При вводе следует разделять значения пробелами, символами табуляции или перевода строки.

`read(v1,v2,...vn);` – аналогочно readln;

Управляющие структуры в языке Pascal

Условный оператор

```
if <условие> then <оператор 1>[else <оператор 2>]
```

Условие – значение типа boolean или логическая операция. Если условие верно, выполняется оператор, или блок операторов, следующий за then, в противном случае выполняется блок операторов после else, если он есть.

Условия могут быть вложенными и в таком случае, любая встретившаяся часть else соответствует ближайшей к ней "сверху" части then.

Оператор выбора одного из вариантов

```
case Выражение of
Вариант1: Оператор1;
Вариант2: Оператор2;
ВариантN: ОператорN;
[else ОператорN1;]
end;
```

Выражение в простейших случаях может быть целочисленным или символьным. В качестве вариантов можно применять:

1. Константное выражение такого же типа, как и выражение после case. Константное выражение отличается от обычного тем, что не содержит переменных и вызовов функций, тем самым оно может быть вычислено на этапе компиляции программы, а не во время выполнения.

2. Интервал, например: 1..5, 'a'..'z'.

3. Список значений или интервалов, например: 1,3,5..8,10,12.

Выполняется оператор case следующим образом: вычисляется выражение после слова case и по порядку проверяется, подходит полученное значение под какой-либо вариант, или нет. Если подходит, то выполняется соответствующий этому варианту оператор, иначе – есть два варианта. Если в операторе case записана часть else, то выполняется оператор после else, если же этой части нет, то не происходит вообще ничего.

Рассмотрим пример. Пусть пользователь вводит целое число от 1 до 10, программа должна приписать к нему слово "ученик" с необходимым окончанием (нулевое, "а" или "ов").

```
program SchoolChildren;
var n: integer;
begin
    write('Число учеников --> ');
    readln(n);
    write(n,' ученик');
    case n of
        2..4: write('а');
        5..10: write('ов');
    end;
    readln;
end.
```

Цикл с параметром (со счетчиком)

for<переменная>:=<нач_значение> to <кон_значение> do <оператор>.

Вместо to возможно слово downto. Рассмотрим такой пример: требуется вывести на экран таблицу квадратов натуральных чисел от 2 до 20.

```
var i: integer;
begin
    for i:=2 to 20 do
    writeln(i,' ',sqr(i));
end.
```

Цикл с предусловием

while <условие> do <оператор>.

Пока условие истинно, выполняется оператор (в этом случае оператор может не выполниться ни разу, т.к. условие проверяется до выполнения). Под оператором здесь понимается либо простой, либо составной оператор (т.е. несколько операторов, заключённых в `begin ... end`).

Цикл с постусловием

`repeat <оператор> until <условие>`

Цикл работает следующим образом: выполняется оператор, затем проверяется условие, если оно пока еще не выполнилось, то оператор выполняется вновь, затем проверяется условие, и т. д. Когда условие, наконец, станет истинным выполнение оператора, расположенного внутри цикла, прекратится, и далее будет выполняться следующий за циклом оператор.

6.3 Экспериментальная часть

6.3.1 Задание на лабораторную работу

Создать приложение со следующим функционалом:

- программа должна выполнять вычисление значений заранее заданных функций;
- вид функции должен выбираться пользователем из меню, выбор осуществляется путем ввода номера функции в списке;
- параметры функции вводятся пользователем с клавиатуры.

Варианты заданий

1. $|-x^3+4x+31|$; $\cos(3x-\pi y/2)^5$ $tg(4y/\pi)$; $x^{3/2}-2y+1$;
2. $\sin(\pi x/2)*\sin(\pi y/2)$; $|x-6x^3-17|$; $ctg(2x)$; $x^2-10y^{1/2}-30$
3. $|-x^4-10x-30|$; $\sin(2x^2-2\,\pi\, y)$; $(x+y)^{1/2}$; $ctg(x/2)$;
4. $\cos(2x/\pi)*\sin(3y-2\pi)$; $|-x^5+3x+2|$; $(x-6y)^{3/2}+9$; $tg(y/2)$

6.3.2 Методические указания по выполнению работы

Вывести пользователю список функций для выбора.

На основе выбранного пользователем пункта меню сделать запрос на ввод одного или двух аргументов функции.

Проанализировать ввод данных пользователем — проверить, существует ли значение выбранной функции в выбранной точке. Если не существует, то вывести соответствующее сообщение, иначе на основе введенных данных произвести вычисление значения функции и вывести это значение на экран.

6.3.3 Содержание отчета

Результаты оформить в виде отчета, где привести:

- текст программы на языке Pascal;
- описание используемых в программе операторов, функций и процедур;
- скриншоты результатов выполнения программы.

Лабораторная работа №7. Сортировка массивов

7.1 Цель работы

Знакомство с типом данных “массив”, получение навыков ввода, обработки массивов, создание диалогового приложения, осуществляющего сортировку массивов различными методами.

7.2 Теоретическая часть

Массив – упорядоченный набор однотипных переменных, объединенных одним именем. В качестве типа элементов массива можно использовать все числовые, символьный, строковый и логический типы. Каждый элемент массива имеет свой номер (индекс). Для индексов массивов подходит любой порядковый тип, то есть такой, который в памяти машины представляется целым числом.

Каждый элемент является переменной, т.е. обладает своим именем и значением.

Массив относится к так называемым структурированным данным, то есть таких, что имеют фиксированную внутреннюю структуру (организацию).

При обращении к отдельному элементу массива необходимо указать его индекс (местонахождение в массиве):

```
A[7]
i:=7;
A[i]
```

Здесь i – индекс элемента массива

Объявление массива

Массивы, как и другие переменные, должны быть объявлены в разделе var

```
var
    Mas: array [1..15] Of real;
    Work: array [(Mon, Tue, Wed)] Of integer;
    B: array ['A'..'Z'] Of boolean;
    C: array [1..3, 1..5] Of real;
```

Ввод массива

Чтобы заполнить массив данными существуен несколько способов:
– непосрерственное присваивание значений элементам;
– генерация и присваивание значений с помощью функции random;
– ввод значений элементов с клавиатуры.
Ввод элементов одномерного массива с клавиатуры:

```
var
    A : array[1..20] of real;
begin
    writeln('Введите элементы массива:');
    for i:=1 to n do readln(A[i]);
...
```

Заполнение массива случайными числами.

В этом случае необходимо перезапустить генератор случайных чисел. Затем в цикле (например, в цикле с параметром, где в качестве параметра выступает индекс массива) сгенерровать значения для всех элементов.

```
randomize;
for
 i:=1
 to n do
        a[i]:=random(100);
...
```

Сортировка массивов

Сортировка массивов – это упорядочение их элементов

Метод пузырька (Bubble Sort)

Суть метода заключается в многократном проходе по списку. На каждом шаге последовательно сравниваются пары соседних элементов, и если порядок в такой паре неверный, то элементы в паре меняются местами. При проходе алгоритма, элемент, стоящий не на своей позиции, «всплывает» до нужной позиции как пузырёк, откуда и название

алгоритма.

Например, чтобы отсортировать массив размером N по возрастанию элементов, необходимо выполнить:

```
для j от 1 до N-1 выполнять
    для i от 1 до N-j выполнять
        если M[i] > M[i+1] то
            обмен M[i] и M[i+1];
```

Обмен между значениями элементов можно осуществить при помощи вспомогательной переменной, например:

```
Tmp := M[i];
M[i] := M[i+1];
M[i+1] := Tmp;
```

Сортировка со вставками (Insertion Sort)

Сортируемый массив просматривается в порядке возрастания номеров и каждый элемент вставляется в уже просмотренную часть массива так, чтобы сохранить порядок.

```
для i от 2 до N do
    нц
        Tmp:=M[i];
        j:=i-1;
        пока (j>0) и (M[j]>Tmp) do
            нц
                M[j+1]:=M[j];
                j:=j-1;
            кц
        M[j+1]:=Tmp;
    кц
```

7.3 Экспериментальная часть

7.3.1 Задание на лабораторную работу

Реализовать пользовательское приложение со следующим функционалом:

– программа должна производить сортировку массива двумя различными способами;

– начальный массив формируется из случайных чисел;

– на экран выводятся начальный и конечный массив;

– выход осуществляется при выборе соотвествующего пункта меню.

7.3.2 Порядок выполнения работы

Объявить массив из 100 элементов а разделе описания переменных.

В основном блоке программы заполнить массив случайными числами в интервале [0,99].

Вывести на экран заполненный массив, расположив элементы в виде таблицы 10x10.

Вывести меню из трех пунктов:

– сортировка методом пузырька;

– сортировка методом вставок;

– выход.

Сделать обработку всех пунктов меню, воспользовавшись оператором case или оператором if и командами языка Pascal.

Перед сортировкой предварительно скопировать массив в другой (для того, чтобы начальный массив сохранился после сортировки). Отсортировать массив по возрастанию элементов и вывести результат на экран в виде таблицы (наименьший элемент должен оказаться в левом верхнем углу экрана, наибольший – в правом нижнем).

После вывода отсортированного массива снова вывести меню на экран.

Процесс повторять до тех пор, пока не будет выбран пункт меню “выход”.

6.3.3 Содержание отчета

Результаты оформить в виде отчета, где привести:

– текст программы на языке Pascal;

– описание используемых в программе типов данных, операторов, функций и процедур;

– скриншоты результатов выполнения программы.

Лабораторная работа №8. Использование подпрограмм в Pascal

8.1 Цель работы

Знакомство с понятиями "процедура" и "функция" в языке Pascal, изучение их сходств и различий. Получение навыков в модульном программировании.

8.2 Теоретическая часть

Существуют случаи, когда в программе приходится выполнять одни и те же вычисления, но при различных исходных данных. повторяющиеся вычисления выделяют в самостоятельную часть программы, которая может быть использована многократно по мере необходимости. Такая автономная часть программы, реализующая определенный алгоритм, оформленная в виде отдельной синтаксической конструкции, снабжённая именем и допускающая обращение к ней из различных частей общей программы, называется подпрограммой.

Подпрограмма – это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем.

Для использования подалгоритма в качестве подпрограммы его необходимо описать в разделе описания подрограмм. Для вызова в основной части программы необходимо вызвать подпрограмму с нужными параметрами Это приведет к выполнению входящих в подпрограмму операторов, работающих с указанными параметрами. После выполнения подпрограммы работа продолжается с той команды, следует за вызовом подпрограммы.

В Pascal существует два вида подпрограмм – процедуры и функции.

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определённой в ней последовательности действий. Процедуры служат для задания совокупности действий, направленных на изменение внешней по отношению к ним программной обстановки.

Функция отличается от процедуры тем, что возвращает результат указанного при её описании типа. Вызов функции может осуществляться из выражения, где имя функции используется в качестве операнда. Функции являются частным случаем процедур, и обязательно возвращают в точку вызова результат как значение имени этой функции. При использовании функций необходимо учитывать совместимость типов в выражениях.

Для обмена информацией между процедурами и функциями и

другими блоками программы существует механизм входных и выходных параметров.

Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные данные для подпрограммы), а выходными – передающиеся из подрограммы в вызывающий блок (результаты работы подпрограммы).

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют формальными, а те, которые используются при ее вызове – фактическими.

Описание и вызов процедур и функций

Описание подпрограммы производится в разделе описаний основной программы. Любая процедура оформляется аналогично программе, может содержать заголовок, разделы описаний и операторов.

Синтаксис заголовка процедуры:

Формат описания процедуры имеет вид:

```
    procedure имя процедуры (формальные параметры);
раздел описаний процедуры
begin
    исполняемая часть процедуры
end;
```

Формат описания функции:

```
function имя функции формальные параметры): тип результата;
раздел описаний функции
begin
    исполняемая часть функции
end;
```

Формальные параметры в заголовке процедур и функций записываются в виде:

```
    var имя праметра: имя типа
```

и отделяются друг от друга точкой с запятой. Ключевое слово var может отсутствовать. Если параметры однотипны, то их имена можно

перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные имена типов, либо имена типов, определенные с помощью команды type. Список формальных параметров может отсутствовать, при этом символ " ; " ставится сразу за именем процедуры и данные из места вызова процедуры в её тело не передаются.

Вызов процедуры производится оператором, имеющим следующий формат:

```
    имя
процедуры(список фактических параметров);
```

В списке фактических параметров параметры перечисляются через запятую в соответсвующем порядке.

Выполнение оператора вызова процедуры состоит в том, что все формальные параметры заменяются соответствующими фактическими. После этого создается динамический экземпляр процедуры, который и выполняется. После выполнения процедуры происходит передача управления в основную программу, т.е. начинает выполняться оператор, следующий за оператором вызова процедуры.

Вызов функции в Pascal может производиться аналогичным способом, кроме того имеется возможность осуществить вызов внутри какого-либо выражения. В частности имя функции может стоять в правой части оператора присваивания, в разделе условий оператора if и т.д.

Для передачи в вызывающий блок выходного значения функции в исполняемой части функции для возврата какого-либо значения необходимо поместить следующую команду:

```
имя функции := результат;
```

При вызове процедур и функций необходимо соблюдать следущие правила:

– количество фактических параметров должно совпадать с количеством формальных;

– соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Следует отметить, что имена формальных и фактических параметров могут совпадать. Это не приводит к проблемам, так как соответствующие им переменные все равно будут различны из-за того, что хранятся в разных областях памяти.

Кроме того, все формальные параметры являются временными переменными – они создаются в момент вызова подпрограммы и уничтожаются в момент выхода из нее.

В отличие от констант и переменных, объявление подпрограммы может быть оторвано от ее описания. В этом случае после объявления нужно указать ключевое слово forward:

```
function<имя_функции>[(<параметры>):<тип_результата>;
forward;
procedure <имя_процедуры> [(<список_параметров>)];
forward;
```

Если объявление подпрограммы было оторвано от ее описания, то описание начинается дополнительной строкой с указанием только имени подпрограммы:

```
function <имя_подпрограммы>;
```

или

```
procedure <имя_подпрограммы>;
```

Описания двух различных подпрограмм не могут пересекаться: каждый блок должен быть логически законченным. Однако внутри любой подпрограммы могут быть описаны другие процедуры или функции – вложенные. На них распространяются все те же правила объявления и описания подпрограмм.

Рассмотрим использование процедуры на примере программы поиска максимума из двух целых чисел.

```
var x,y,m,n: integer;
procedure MaxNumber(a,b: integer; var max: integer);
begin
    if a>b then max:=a else max:=b;
end;
begin
    write('Введите x,y ');
    readln(x,y);
    MaxNumber(x,y,m);
    MaxNumber(2,x+y,n);
    writeln('m=',m,'n=',n);
end.
```

Аналогичную задачу можно решить с помощью функции:

```
var x,y,m,n: integer;
function MaxNumber(a,b: integer): integer;
```

```
var max: integer;
begin
    if a>b then max:=a else max:=b;
    MaxNumber := max;
end;
begin
    write('Введите x,y ');
    readln(x,y);
    m := MaxNumber(x,y);
    n := MaxNumber(2,x+y);
    writeln('m=',m,'n=',n);
end.
```

Передача параметров

В стандарте языка Паскаль передача параметров может производиться двумя способами – по значению и по ссылке. Параметры, передаваемые по значению, называют параметрами-значениями, передаваемые по ссылке – параметрами-переменными. Последние отличаются тем, что в заголовке процедуры (функции) перед ними ставится служебное слово var.

При первом способе (передача по значению) значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения процедуры (функции) исходные данные (фактические параметры) измениться не могут. Поэтому таким способом передают данные только из вызывающего блока в подпрограмму (т.е. входные параметры). При этом в качестве фактических параметров можно использовать и константы, и переменные, и выражения.

При втором способе (передача по ссылке) все изменения, происходящие в теле процедуры (функции) с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров.

Изменения происходят с переменными вызывающего блока, поэтому по ссылке передаются выходные параметры. При вызове соответствующие им фактические параметры могут быть только переменными.

Выбор способа передачи параметров при создании процедуры (функции) : входные параметры нужно передавать по значению, а выходные – по ссылке.

Практически это сводится к расстановке в заголовке процедуры (функции) описателя var при всех параметрах, которые обозначают результат работы подпрограммы. Однако, в связи с тем, что функция возвращает только один результат, в ее заголовке использовать параметры-переменные не рекомендуется.

Локальные и глобальные идентификаторы

Использование процедур и функций в Паскале тесно связано с некоторыми особенностями работы с идентификаторами (именами) в программе. В частности, не все имена всегда доступны для использования. Доступ к идентификатору в конкретный момент времени определяется тем, в каком блоке он описан.

Имена, описанные в заголовке или разделе описаний процедуры или функции называют локальными для этого блока. Имена, описанные в блоке, соответствующем всей программе, называют глобальными. Следует помнить, что формальные параметры прцедур и функций всегда являются локальными переменными для соответствующих блоков.

Основные правила работы с глобальными и локальными именами можно сформулировать так:

– локальные имена доступны (считаются известными, "видимыми") только внутри того блока, где они описаны. Сам этот блок, и все другие, вложенные в него, называют областью видимости для этих локальных имен;

– имена, описанные в одном блоке, могут совпадать с именами из других, как содержащих данный блок, так и вложенных в него. Это объясняется тем, что переменные, описанные в разных блоках (даже если они имеют одинаковые имена), хранятся в разных областях оперативной памяти.

Глобальные имена хранятся в области памяти, называемой сегментом данных (статическим сегментом) программы. Они создаются на этапе компиляции и действительны на все время работы программы.

В отличие от них, локальные переменные хранятся в специальной области памяти, которая называется стек. Они являются временными, так как создаются в момент входа в подпрограмму и уничтожаются при выходе из нее.

Имя, описанное в блоке, "закрывает" совпадающие с ним имена из блоков, содержащие данный. Это означает, что если в двух блоках, один из которых содержится внутри другого, есть переменные с одинаковыми именами, то после входа во вложенный блок работа будет идти с локальной для данного блока переменной. Пременная с тем же имнем, описанная в объемлющем блоке, становится временно недоступной и это продолжается до момента выхода из вложенного блока.

Рекомендуется все имена, которые имеют в подпрограммах чисто внутреннее, вспомогательное назначение, делать локальными. Это предохраняет от изменений глобальные объекты с такими же именами.

8.3 Экспериментальная часть
8.3.1 Задание на лабораторную работу

Создать программу, в которой производится вычисление заранее заданной преподавателем функции от заранее заданного набора аргументов и осуществляется вывод результатов.

Расчет функции должен быть произведен с помощью механизма функций языка Pascal. Вывод результатов должен производиться на консоль пользователя с помощью специально написанной процедуры.

Варианты функций

1. cos(3x-πx/2)5 ; аргументы: x изменяется от 0 до 2π с шагом π/10

2. cos(2x/π)*sin(3x-2π); аргументы: x изменяется от π/2 до 3π/2 с шагом π/20

3. lg x* (2x2-1); аргументы: x изменяется от 1 до 20 с шагом 1

Процедура должна осуществлять вывод аргументов и соответствующих им значений функций в следующем формате:

x=<значение x, два знака после запятой> f(x)=<значение функции, четыре знака после запятой>

Результаты работы оформить в виде отчета, где привести:

– текст программы на языке Pascal;

– описание используемых в программе типов данных, операторов, функций и процедур;

– скриншоты результатов выполнения программы.

Лабораторная работа №9. Файловый ввод вывод в программах на языке Pascal

9.1 Цель работы

Изучение основных функций работы с файлами в Pascal, организация ввода и вывода структурированных данных из файлов.

9.2 Теоретическая часть

Записи

Тип “запись” является структурированным типом данных, то есть таким, переменные которого составлены из нескольких частей. В Pascal существует возможность объединить в одну переменную данные разных типов (тогда как в массиве все элементы имеют одинаковый тип).

Пусть в переменной требуется хранить сведения о некотором человеке:

ФИО, пол, адрес, телефон. Тогда для хранения этих данных будет удобен такой тип:

```
type tPerson = record
    Name,Surname,SecondName: string[30];
    Male: boolean;
    Address: string[50];
    Phone: string[11];
end;
```

Объявление переменной типа запись выполняется стандартно, в разделе var.

Части записи (в нашем случае: Name, Surname, SecondName, Male, Address, Phone) называются полями. Обращение к полю записи в программе производится с помощью знака '.' (точка). Пример обращения к полям:

```
var emp: tPerson;
...
begin
    ...
    emp.Surname:='Иванов';
    emp.Name:='Иван';
    emp.SecondName:='Иванович';
    ...
```

В случаях, когда приходится много раз обращаться к полям одной и той же записи, можно воспользоваться ключевым оператором with, который упрощает ссылку к структурированным переменным:

```
with <имя_записи> do <оператор>;
```

Пример:

```
with emp do
begin
     Surname:=' Иванов';
     Name:='Иван';
     SecondName:='Иванович';
     ...
end;
```

Записи можно включать в состав более сложных переменных, например массивов и других записей:

```
type tStaff = array [1..30] of tPerson;
```

Работа с файлами

При работе с файлами существует определенный порядок действий, которого необходимо придерживаться. Вот все эти действия:

1) создание (описание) файловой переменной;
2) связывание этой переменной с конкретным файлом на диске или с устройством ввода-вывода (экран, клавиатура, принтер и т.п.);
3) открытие файла для записи либо чтения;
4) действия с файлом: чтение либо запись;
5) закрытие файла.

Типы файловых переменных

text – текстовый файл. Из переменной такого типа мы сможем читать строки и символы.

file of _любой_тип_ – так называемые "типизированные" файлы, то есть файлы, имеющие тип. Этот тип определяет, какого рода информация содержится в файле и задается в параметре_любой_тип_. Например:

```
F: file of integer;
```

Файл F содержит числа типа integer; Соответсвенно, читать из такого файла можно только переменные типа integer, ровно как и писать.

```
type
    A = record
        I,J: Integer;
        S: String[20];
    end;
var
    F: File of A;
```

file – нетипизированный файл:

```
    F: File;
```

Чтение и запись в типизированные файлы отличается от работы с файлами других типов. Эти действия производятся путем указания количества байт, которые нужно прочитать, а также указанием области памяти, в которую нужно прочитать эти данные.

Связывание переменной с файлом

Выполняется одинаково для всех типов файлов:

```
assign(<переменная_файлового_типа>,'<путь к файлу>');
```

В качестве параметров задаются переменная любого файлового типа и строка – путь к файлу:

```
var
    F1: file of integer;
begin
    Assign(F1, 'int.txt');
```

Открытие файла

При открытии файла необходимо учитывать, зачем открывается файл – для записи или чтения. Более того, в зависимости от типа файла процедуры выполняют различные действия.

```
    reset(<любая_файловая_переменная>);
```

Открывает файл на чтение. В качестве параметра – файловая

переменная любого типа. В случае с текстовым файлом, он открывается только на чтение. В случае с типизированным и нетипизированным файлом – открывается на чтение и запись.

```
append(T: Text);
```

Открывает текстовый файл (только текстовый!) на запись. Reset при задании параметра типа Text не позволит писать в него данные, открыв файл лишь для чтения. Для записи в текстовый файл нужно использовать append. Если чтение – reset.

Если файл открыт на чтение, не нужно закрывать его и открывать снова на запись. В этом случае файл закрывается сам и открывается заново. При записи данных в файл при открытии его с помощью этой процедуры они записываются в конец файла.

```
rewrite(F);
```

Создает новый файл либо перезаписывает существующий. Необходимо быть осторожным при использовании этой процедуры, т.к. файл, открытый таким образом будет полностью перезаписан.

Закрытие файла

Закрытие файла производиться с помощью процедуры Close(F), где F – это переменная файлового типа. Эта процедура одна для всех типов файлов.

Запись и чтение файлов

Текстовые и типизированные файлы

Чтение файлов

Чтение файлов производится с помощью процедур read и readln. Они используются также, как и при чтении информации с клавитуры. Отличие лишь в том, что перед переменной, в которую помещается считанное значение, указывается переменная файлового типа (дескриптор файла):

```
read(F, C);
```

где F – дескриптор файла, C – переменная (char, string – для текстовых, любого типа – для типизированных файлов).

Запись в файлы.

Запись в файлы производиться точно так же, как и запись на экран с помощью процедур write и writeln. Как и в случае с чтением, перед записываемой в файл переменной указывается дескриптор файла:

```
write(F, S);
```

где F – дескриптор, S – переменная.

При этом переменная должна соответствовать типу файла.

```
program cat;
var
    f: text;
    c: char;
begin
    assign(f,'prog.pas');
    reset(f);
    while not eof(f) do
    begin
        while not eoln(f) do
        begin
            read(f,c);
            write(c);
        end;
        readln(f);
        writeln;
    end;
    close(f);
end.
```

В программе бывает необходимо определить, дошёл ли указатель файла до конца строки или до конца файла. В этом случае полезно использовать такие функции:

```
eoln(TxtFile: text): boolean;

eof(TxtFile: text): boolean;
```

Первая принимает значение true (истина), если указатель стоит на конце строки, вторая – то же самое для конца файла.

Нетипизированные файлы

Суть таких файлов заключается в следующем: имея файл без определенного типа, мы можем читать из него любые данные, будь то строки, символы или записи.

Читая данные из файла без типа мы получаем блоки информации, которые составляют обычный набор байт. Указывая переменную, в которую эти байты надо поместить, мы как бы "на ходу преобразуем" эти данные к нужному типу.

Чтение из файлов без типа.

Сама процедура связывания файловой переменной с внешним файлом и его открытие ничем чем отличаются от обычного порядка действий. Только переменная в данном случае должна иметь тип File; , то есть быть файлом без типа.

Чтение происводиться с помощью процедуры blockread:

```
blockread(F: file, Buf: var, size: word, result:word)
```

F: file; – переменная типа file, именно из этой переменной и происходит чтение данных. Buf: var; – переменная любого типа. В эту переменную помещаются прочитанные данные., size: word; – количество считываемых байт, result: word; – в эту переменную помещается реальное количество байт, которые были прочитаны.

Процедура работает следующим образом: из файла F считывается Size записей, которые помещаются в память, начиная с первого байта переменной Buf.

После выполнения процедуры реальное количество прочитанных байт помещается в переменную Result. Здесь надо сказать, что эта переменная совсем не обязательно должна присутствовать в качестве параметра, то есть ее попросту можно опустить. Однако иногда она довольно полезна и даже необходима – например, если чтение было окончено до того, как было прочитано требуемое количество байт (достигнут конец файла), мы можем это отследить через переменную Result. Если же в этом случае (чтение данных после конца файла) переменная Result не будет указана, то образуется ошибка времени выполнения

```
N100 "Disk read error" (Runtime error 100).
```

Запись в файлы без типа.

```
BlockWrite(F:File, Buf: Var, Size: Word, Result:Word)
```

F: File; – переменная типа File; Buf: Var; – переменная любого типа.

Начиная с этой переменной, данные будут записываться в файл. Size: Word; – количество записываемого блока данных в байтах. Result: Word; – в эту переменную помещается реальное количество байт, которые были записаны.

```
var
    Fin, fout : file;
    NumRead,NumWritten : word;
    Buf : Array[1..2048] of byte;
    Total : longint;
begin
    Assign (Fin, Paramstr(1));
    Assign (Fout,Paramstr(2));
    Reset (Fin,1);
    Rewrite (Fout,1);
    Total:=0;
    Repeat
        BlockRead (Fin,buf,Sizeof(buf),NumRead);
        BlockWrite (Fout,Buf,NumRead,NumWritten);
        inc(Total,NumWritten);
    Until (NumRead=0) or (NumWritten<>NumRead);
    Write('Copied',Total,'bytes from file',paramstr(1));
    Writeln (' to file ',paramstr(2));
    close(fin);
    close(fout);
end.
```

9.3 Экспериментальная часть

9.3.1 Задание на лабораторную работу

Написать программу, осуществляющую ввод структурированных данных в файл и вывод данных из этого файла на экран (сделать хранилище данных в файле).

Варианты задания

1. Карточка студента
2. Библиотека

3. Фонотека
4. Телефонный справочник
5. Отдел кадров
6. Страны мира

9.3.2 Порядок выполнения работы

Создать новый тип данных «запись» согласно полученному варианту.

Вывести приглашение пользователю на ввод данных об объекте.

Заполнить поля записи введенными пользователями данными.

Записать в файл, открыв его в соответствующем режиме. Процесс записи оформить в виде цикла, на каждом шаге которого вводятся данные об очередном объекте и выводится приглашение “Ввести данные о новом объекте? (Yes/No)”. Выход из цикла – ввод пользователем “n”(No) после ввода данных об очередном объекте.

Считать все записи из файла и вывести их на экран.

Лабораторная работа №10. Алгоритмы на списках

10.1 Цель работы

Знакомство со ссылочными реализациями структур данных. Создание приложения, осуществляющего работу со списками.

10.2 Теоретическая часть

Списки

Классический пример структуры данных последовательного доступа, в которой можно удалять и добавлять элементы внутри структуры, — это линейный список.

Список – это конечное множество динамических элементов, размещающихся в разных областях памяти и объединенных в логически упорядоченную последовательность с помощью специальных указателей (адресов связи).

Список – структура данных, в которой каждый элемент имеет информационное поле (поля) и ссылку (ссылки), то есть адрес (адреса), на другой элемент (элементы) списка. Список - это так называемая линейная структура данных, с помощью которой задаются одномерные отношения.

Каждый элемент списка содержит информационную и ссылочную части. Порядок расположения информационных и ссылочных полей в элементе при его описании - по выбору программиста, то есть фактически произволен.

Информационная часть в общем случае может быть неоднородной, то есть содержать поля с информацией различных типов. Ссылки однотипны, но число их может быть различным в зависимости от типа списка. В связи с этим для описания элемента списка подходит только тип «запись», так как только этот тип данных может иметь разнотипные поля. Например, для однонаправленного списка элемент должен содержать как минимум два поля: одно поле типа «указатель», другое - для хранения данных пользователя. Для двунаправленного – три поля, два из которых должны быть типа «указатель».

Описать элемент однонаправленного списка можно следующим образом:

```
type
    el=^zap;
    zap=record
        inf1 : integer; { первое информационное поле
}
    inf2 : string;
```

```
 { второе информационное поле
 }
        next : el;
 {ссылочное поле }
    end;
```

Из этого описания видно, что имеет место рекурсивная ссылка: для описания типа point используется тип zap, а при описании типа zap используется тип point.

По соглашениям Паскаля в этом случае сначала описывается тип «указатель», а затем уже тип связанной с ним переменной. Правила Паскаля только при описании ссылок допускают использование идентификатора (zap) до его описания. Во всех остальных случаях, прежде чем упомянуть идентификатор, необходимо его определить.

```
var
first,
{ указатель на первый элемент списка }
p, q , t : el;
{ рабочие указатели, с помощью которых будет
выполняться работа с элементами списка }
```

Формирование пустого списка

```
procedure Create_Empty_List ( var first : el);
begin
    first = nil;
End;
```

Формирование очередного элемента списка

```
procedure Create_New_Elem(var p: el);
begin
    New (p);
    Writeln      ('введите      значение      первого
информационного поля: ');
    Readln ( p^.inf1 );
    Writeln      ('введите      значение      второго
информационного поля: ');
    Readln ( p^.inf2 );
    p^.next  :=  nil;  {все  поля  элемента  должны  быть
инициализированы}
end;
```

Подсчет числа элементов списка

```
function Count_el(First:el):integer;
var
    K : integer;
    q : el;
begin
    If First = Nil then
        k:=0 { список пуст }
    Else
    begin {список существует}
        k:=1; {в списке есть хотя бы один элемент}
        q:=First;    {перебор    элементов    списка
начинается с первого}
        while q^.Next <> Nil do
        begin
            k:=k+1;
            q:=q^.Next;    {переход    к    следующему
элементу списка}
        end;
    end;
    Count_el:=k;
end;
```

Включение элемента в конец списка

```
procedure Ins_end_list(P : el; Var First : el);
begin
    If First = Nil Then
        First:=p
    Else
    Begin
        q:=First;  {цикл  поиска  адреса  последнего
элемента}
        While q^.Next <> Nil do
            q:=q^.Next;
        q^.Next:=p;{ссылка  с  бывшего  последнего  на
включаемый элемент}
        P^.Next:=Nil; {не обязательно}
    End;
end;
```

Включение в середину (после i-ого элемента)

```
procedure Ins_after_I ( first : el; p : el; i : integer);
var
    t, q : el;
    K ,n : integer;
begin
    n   :=   count_el(first);   {определение   числа элементов списка}
    if (i < 1 ) or ( i > n )then
    begin
        writeln ('i задано некорректно');
        exit;
    end
    else
    begin
        if i = 1 then
        begin
            t := first;{адрес 1 элемента}
            q := t^.next; {адрес 2 элемента}
            t^.next := p;
            p^.next := q;
        end
        else
            if i = n then
            begin   {   см.   случай   вставки   после последнего элемента}
                . . .
            end
            else {вставка в «середину» списка}
            begin
                t := first;
                k := 1;
                while ( k < i ) do
                begin {поиск адреса i-го элемента}
                    k := k + 1;
                    t := t^.next;
                end;
                q := t^.next;
                {найдены адреса i-го (t) и i+1 -го (q) элементов }
            t^.next := p;
                p^.next := q;
```

```
                {элемент с адресом p вставлен}
            end;
        end;
end;
```

Удаление элемента из середины списка (i-ого элемента)

```
Procedure Del_I_elem ( first : el; i : integer);
Var
    t, q, r : el;
    K ,n : integer;
Begin
    n   :=   count_el(first);   {определение   числа
элементов списка}
    if (i < 1 ) or ( i > n ) then
    begin
        writeln ('i задано некорректно');
        exit;
    end
    else
    begin {нужно добавить подтверждение удаления }
        if i = 1 then
        begin {удаляется 1 элемент}
            t := first;
            first:= first^.next;
            dispose ( t);
        end
        else
            if i = n then
            begin  {  см.  случай  удаления  последнего
элемента}
            . . .
            end
            else {удаление из «середины» списка}
            begin
                t := first;
                q := nil;
                k := 1;
                while ( k < i ) do
                begin {поиск адресов (i-1)-го и i-го
элементов}
                    k := k + 1;
                    q := t;
                    t := t^.next;
```

```
                end;
                r := t^.next;
{найдены адреса i-го (t), (i-1)-го (q) и (i+1)-го (r)
элементов
}
                q^.next := r;
                dispose ( t );
                {удален i-ый элемент }
            end;
        end;
end;
```

Удаление всего списка с освобождением памяти

```
procedure Delete_List(Var First : el);
var
    P, q : el;
    Answer : string;
begin
    If First <> Nil Then
    begin { список не пуст }
        writeln ( ' Вы хотите удалить весь список ?
(да/нет)' );
        readln ( answer );
        if answer = 'да' then
        begin
            q:=First;
            p:=nil;
            while ( q <> nil ) do
            begin
                p:=q;
                q:=q^.Next;
                Dispose(p);
            end;
            First:=Nil;
        end;
    end
    else
        writeln ('список пуст ');
end;
```

В рамках данной лабораторной работы, студентам предлагается реализовать однонаправленный список.

10.3 Экспериментапльная часть
10.3.1 Задание на лабораторную работу

Создать приложение со следующим функционалом:

– приложение предназначено для создания и редактирования однонаправленных списков;

– выбор операции над списком производится пользователем в режиме меню.

– ввод параметров производится с клавиатуры.

10.3.2 Порядок выполнения лабораторной работы

Вывести приглашение пользователю на ввод выбор операции над списком:

1) создание нового списка;
2) добавление нового звена в список;удаление звена из списка;
3) удаление списка;
4) выход.

При выборе первого пункта вывести приглашение на ввод количества элементов списка. Вывести приглашение на ввод значений элементов. Сохранить введенные данные в переменные типа «структура». Вывести содержимое списка на экран.

При выборе второго пункта вывести приглашение на ввод информационного поля нового элемента, а также номера звена, после которого нужно вставить новое звено. Создать новое звено. Вывести содержимое списка на экран.

При выборе третьего пункта вывести приглашение на ввод звена списка, которое нужно удалить. Произвести удаление звена. Вывести содержимое списка на экран.

При выборе четвертого пункта меню осуществить очистку памяти

Лабораторная работа №11. Сортировка списков

11.1 Цель работы

Закрепление навыков работы со ссылочными реализациями структур данных. Создание приложения, осуществляющего ввод, редактирование, сортировку и удаление однонаправленного списка.

11.2 Экспериментальная часть

11.2.1 Задание на лабораторную работу

Создать приложение со следующим функционалом:

– приложение предназначено для создания и редактирования однонаправленных списков;

– выбор операции над списком производится пользователем в режиме меню;

– ввод данных производится с клавиатуры;

– сортировка производится по строковому полю методом пузырька.

Варианты заданий

Список организовать согласно варианту из предыдущей лабораторной работы

11.2.2 Порядок выполнения работы

Преобразовать программу из лабораторной работы No10 согласно варианту задания. Добавить процедуру сортировки списка.

В основной части программы вывести приглашение пользователю на ввод выбор операции над списком:

– создание нового списка;

– сортировка списка;

– добавление нового звена в список;

– удаление звена из списка;

– удаление списка;

– выход.

Сделать обработчик выбора пунктов меню с вызовом соответствующих процедур и функций работы со списком.

Лабораторная работа №12. Введение в объектно-ориентированное программирование. Наследование

12.1 Цель работы

Знакомство с основными принципами объектно-ориентированного программирования и синтаксисом языка Pascal для создания объектно-ориентированных приложений. Практическое применение принципа наследования – построение иерархической схемы классов.

12.2 Теоретическая часть

В языке Pascal основным элементом объектно-ориентированных приложений является объект. Синтаксис описания объекта:

```
type
    <имя объекта> = object
    <список членов(поля и заголовки методов)>;
end;
```

Члены класса можно разделить на две группы.

1. Поля данных – данные, определяющие состояние объекта
2. Методы – процедуры и функции, которые могут использовать поля объекта и внешние данные.

Каждый член класс имеет атрибут доступа, при помощи которого определяется “зона видимости” для члена класса. Всего таких атрибутов три:

public – член объекта может использоваться любой функцией(процедурой);

private – член объекта может использоваться только функциями-членами объекта;

protected – член объекта может использоваться функциями-членами объекта, а также членами объектов, для которого данный объект является базовым(предком).

Основные принципы ООП

Инкапсуляция(encapsulation). Комбинирование записей с процедурами и функциями, манипулирующими полями этих записей, формирует новый тип данных – объект.

Инкапсуляция производится таким образом, чтобы пользователь объекта мог видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов объекта и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в

классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов.

Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между объектами и, соответственно, упростить независимую реализацию и модификацию классов.

Наследование(inheritance). Наследованием называется возможность порождать один объект от другого с сохранением всех свойств и методов объекта-предка (прародителя) и добавляя, при необходимости, новые свойства и методы. Набор объектов,связанных отношением наследования, называют иерархией.

Наследование призвано отобразить такое свойство реального мира, как иерархичность.

Важно помнить то, что если характеристика однажды определена на каком-то уровне иерархии, то все объекты, расположенные ниже данного уровня, содержат эту характеристику.

Полиморфизм(polymorphism). Присваивание действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причем каждый объект иерархии выполняет это действие способом, именно ему подходящим.

Экземпляры объектных типов

Экземпляры объектных типов описываются в точности так же, как в Паскале описывается любая переменная, либо статическая, либо указатель, ссылающийся на размещенную в динамической памяти переменную:

```
var
    Emp: TEmployee;
```

Поля объектов

К полю объекта можно обратиться так же, как к полю обычной записи, либо с помощью оператора with, либо путем уточнения имени с помощью точки.

Например:

```
Emp.Rate := 10;
with Emp do
begin
    Name := 'Ivanov';
    Title := 'programmer';
end;
```

Методы

Внутри объекта метод определяется заголовком процедуры или функции, действующей как метод:

```
type
    TEmployee = object
    Name, Title: string[25];
    Rate: Real;
    procedure Init (AName,
 ATitle:
 String;
 ARate:
 Real);
end;
```

Примечание: Поля данных должны быть описаны перед первым описанием метода.

При определении метода после описания объекта имени метода должно предшествовать имя типа объекта, которому принадлежит этот метод, с последующей точкой:

```
procedure
 TEmployee.Init(AName,
 ATitle:
 string; ARate:
Real);
begin
    Name := AName;
    Title := ATitle;
    Rate := ARate;
end;
```

Создание объекта-наследника

```
<имя объекта> = object(<имя обекта-родителя>)
    <список членов>;
end;
```

При создании объекта-наследника, он наследует все поля и методы базового объекта, описанные атрибутами public и protected. Для этого класса нет необходмсоти снова определять эти поля и методы.

12.3 Экспериментальная часть
12.3.1 Задание на лабораторную работу

Создать приложение, реализующее следующую иерархию объектов

TPerson

/
\

TStudent
TTeacher

Объекты TStudent и TTeacher должны наследовать поля и методы объекта TPerson.

12.3.2 Порядок выполнения работы

Создать объект типа TPerson со следующими полями: фамилия, имя, отчество, дата рождения – и двумя методами, выводящими на экран ФИО (фамилию, имя, отчество) и дату рождения.

Создать объект TStudent – потомок класса TPerson, для которого определить новые поля (год зачисления в вуз, номер группы) и методы, выводящие эти поля на экран.

Создать объект

TTeacher – потомок класса

TPerson, для которого определить новое поле (должность) и метод, выводящие это поле.

Создать одну переменную типа TPerson, две переменные типа TStudent и одну переменную типа TTeacher.

В основном блоке программы заполнить все поля всех экземпляров объектов (информацию пользователь вводит с клавиатуры) и вызвать методы.

12.3.3 Содержание отчета

Результат работы оформить в виде отчета, в котором обязательно привести:

- описание всех объектов, их полей и методов;
- скриншоты программы;
- листинг программы.

Лабораторная работа №13. Введение в объектно-ориентированное программирование. Конструкторы и деструкторы

13.1 Теоретическая часть

Существуют специальные методы объектов, которые отвечают за создание экземпляров объекта и их удаление. Это так называемые конструкторы и деструкторы. Конструкторы – методы, основная цель которых заключается в инициализации объекта и распределении памяти для хранения объекта. Как и любой другой метод, конструктор может иметь или не иметь параметров. Конструктор без параметров называется конструктором по умолчанию.

Деструктор разрушает созданный экземпляр. По своей форме конструкторы и деструкторы являются процедурами, но объявляются с помощью зарезервированных слов constructor и destructor.

Объекты могут размещаться в динамической памяти и ими можно манипулировать с помощью указателей. Паскаль включает несколько мощных расширений для выполнения динамического размещения и удаления объектов более легкими и более эффективными способами.

Free Pascal поддерживает расширенный синтаксис процедур new и dispose. В случае, когда нужно выделить память под динамическую переменную объектного типа, при вызове процедуры new может быть указано имя конструктора объекта:

```
Type
    TObj = object;
        Constructor Init
        Destructor Destroy;
        ...
    end;
    Pobj = ^TObj;
Var PP : Pobj;
```

Следующие 3 вызова эквивалентны:

```
pp := new (Pobj,Init);
new(pp,Init);
new (pp);
```

Таким же образом, для выполнения освобождения памяти деструктор можно вызывать как часть расширенного синтаксиса процедуры dispose:

```
dispose(pp, Destroy);
```

13.2 Экспериментальная часть

13.2.1 Задание на лабораторную работу

На основе лабораторной работы No 5 создать приложение для работы с объектами типа “список”: создание объекта в динамической памяти, добавление и удаление элементов из списка, уничтожение объекта из динамической памяти.

13.2.2. Порядок выполнения работы

Создать объект типа TList со следующими обязательными методами: Init(конструктор объекта), Print(вывести список на экран), Add_Item(включение нового элемента в список), Delete_Item(удаление элемента из списка), Destroy(деструктор объекта) – и обязательным полем First, в котором хранится указатель на первый элемент.

В основном блоке программы в динамической памяти создать экземпляр объекта. Осуществить редактирование списка в форме диалога с пользователем (интерфейс должен быть эргономичным!!!). При выходе из программы очистить память, занимаемую списком.

13.2.3 Содержание отчета

Результат работы оформить в виде отчета, в котором кроме обязательных пунктов (титульный лист, цель работы, задание на работу с вариантом, заключение и листинг программы) обязательно привести:

- подробное описание всех объектов (назначение), их полей (назначение и тип данных) и методов (назначение и расширенный список формальных параметров);
- описание использования конструктора и деструктора объекта;
- скриншоты программы(на каждую операцию со списком).

Список рекомендуемой литературы

1. 1 Информатика. Базовый курс : учебное пособие для вузов / ред. С. В. Симонович. - 2-е изд. - СПб. : Питер, 2009

2. Лабораторный практикум по информатике : Учебное пособие для вузов/ В. С. Микшина, Г. А. Еремеева, К. И. Бушмелева и др; Ред. В. А. Острейковский. -М.: Высшая школа, 2003.-375 с.

3. PASCAL 7.0. Практическое программирование. Решение типовых задач: учебное пособие/ Лала Михайловна Климова. - 3-е изд., доп.. - М.: КУДИЦ-ОБРАЗ, 2002. - 516 с.

4. Офицеров Д.В. и др. Программирование на персональных ЭВМ. Практикум. -Минск, Высшая школа. 1993. -256 с.

Приложение А
Образец титульного листа отчета

Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления
и радиоэлектроники»

Кафедра Электронных приборов (ЭП)

Дисциплина «Информатика»

ОТЧЕТ
по лабораторной работе
«_____»

Выполнил студент гр. 348
_____И.О.
Фамилия
«_____» _____
20__ г

Проверил
_____
должность
_____И.О.
Фамилия
«_____»
_____20__ г

20___

Учебное пособие

Шандаров Е.С.

Компьютерный лабораторный практикум

Методические указания к лабораторным работам
по дисциплине «Информатика»

Усл. печ. л. _____ Препринт
Томский государственный университет
систем управления и радиоэлектроники
634050, г.Томск, пр.Ленина, 40