

*Направление подготовки 13.04.02 «Электроэнергетика и электротехника»
Магистерская программа: « Электроэнергетические системы, сети,
электропередачи, их режимы, устойчивость и надежность»
Методическое обеспечение РПД Б1.О.05 «Компьютерные, сетевые и информаци-
онные технологии»*



**Филиал федерального государственного бюджетного образовательного учреждения
высшего образования
«Национальный исследовательский университет «МЭИ»
в г. Смоленске**

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБЕСПЕЧЕНИЯ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА

Направление подготовки (специальность): 13.04.02 «Электроэнергетика и электротехника»

Магистерская программа: «Электроэнергетические системы, сети, электропередачи, их режимы, устойчивость и надежность»

Уровень высшего образования: магистратура

Нормативный срок обучения: 2 года

Форма обучения: очная

Год набора: 2022

*Направление подготовки 13.04.02 «Электроэнергетика и электротехника»
Магистерская программа: «Электроэнергетические системы, сети,
электропередачи, их режимы, устойчивость и надежность»
Методическое обеспечение РПД Б1.О.05 «Компьютерные, сетевые и информаци-
онные технологии»*



**Филиал федерального государственного бюджетного образовательного учреждения
высшего образования
«Национальный исследовательский университет «МЭИ»
в г. Смоленске**

**Методические рекомендации к практическим занятиям
по дисциплине**

КОМПЬЮТЕРНЫЕ, СЕТЕВЫЕ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

(НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ)

Смоленск

1. Цели и задачи, объем практических занятий по дисциплине

Цель практических занятий по дисциплине «Компьютерные, сетевые и информационные технологии» – ознакомление с теоритическими знаниями и основами их применения на практике необходимыми для выполнения лабораторных работ.

2. Задания на практические занятия по дисциплине

Практические занятия по дисциплине проводятся по следующим основным тематикам:

Практическое занятие 1. Этапы развития вычислительной техники. Современный этап развития вычислительной техники.

Практическое занятие 2. Другие математические пакеты, преимущества и недостатки среды Matlab. Основные принципы и методы работы в среде MatLab. Программы MATLAB двух типов — функции и скрипты. Векторы и матрицы

Практическое занятие 3. Ознакомление с программной средой для математических расчетов Mathcad, изучить основные приемы работы в Mathcad для вычисления результатов элементарных математических операций, задания переменных и функций, построения графиков функций, проведения матричных расчетов, решения систем линейных уравнений и нахождения интегралов, производных, пределов..

Практическое занятие 4. Ознакомление с возможностями символьного процессора Mathcad, получение навыка использования режима программирования в Mathcad.

Практическое занятие 5. Ознакомление с программной средой для математических расчетов MATLAB, освоить синтаксис и семантику языка программирования MATLAB.

Практическое занятие 6. Ознакомление с некоторыми возможностями MATLAB по обработке изображений.

Практическое занятие 7. Ознакомление с некоторыми возможностями MATLAB по цифровой обработке звука.

Практическое занятие 8. Моделирование движения заряженной частицы в магнитном поле.

Практическое занятие 9. Подведение итогов проведенной работы.

Конкретное задание для каждого практического занятия хранится у преподавателя и выдается на соответствующем практическом занятии.

3. Технология проведения практических занятий

Для студенческой группы на занятии осуществляется введение в теоритические вопросы, связанные с темой практического занятия. После ознакомления с теорией приводятся практические примеры применения этих знаний и способы реализации изученных программных алгоритмов в различных математических средах.

*Направление подготовки 13.04.02 «Электроэнергетика и электротехника»
Магистерская программа: «Электроэнергетические системы, сети,
электропередачи, их режимы, устойчивость и надежность»
Методическое обеспечение РПД Б1.О.05 «Компьютерные, сетевые и информаци-
онные технологии»*



**Филиал федерального государственного бюджетного образовательного учреждения
высшего образования
«Национальный исследовательский университет «МЭИ»
в г. Смоленске**

**Методические рекомендации к самостоятельной работе студентов
по дисциплине**

КОМПЬЮТЕРНЫЕ, СЕТЕВЫЕ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

(НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ)

Смоленск

1. Общие сведения о самостоятельной работе студентов по дисциплине

Цель самостоятельной работы студента – осмысленно и самостоятельно работать сначала с учебным материалом, получаемым при контактной работе с преподавателем, заложить основы самоорганизации и самовоспитания с тем, чтобы привить умение в дальнейшем непрерывно повышать свою профессиональную квалификацию.

Самостоятельная работа студентов по дисциплине «Компьютерные, сетевые и информационные технологии» проводится в соответствии с рабочей программой дисциплины.

Виды занятий, по которым предусматривается самостоятельная работа студентов, сведены в табл. 1.

Таблица 1

| <i>Вид работ</i> |
|---|
| Подготовка к практическим занятиям (пз) |
| Подготовка к выполнению и защите лабораторных работ (лаб) |
| Подготовка к зачету |

2. Содержание самостоятельной работы студентов по дисциплине

Процесс освоения студентами дисциплины «Компьютерные, сетевые и информационные технологии» включает:

По теме 1 «Введение в историю развития вычислительной техники и информационных технологий» в качестве самостоятельной работы предусматриваются:

- а) подготовка к практическим занятиям,
- б) подготовка к выполнению и защите лабораторных работ № 1 – № 2 (изучение методических указаний),

Рекомендации студентам по самостоятельной работе к теме №1.

- а) подготовка к выполнению и защите лабораторных работ № 1 - 2№

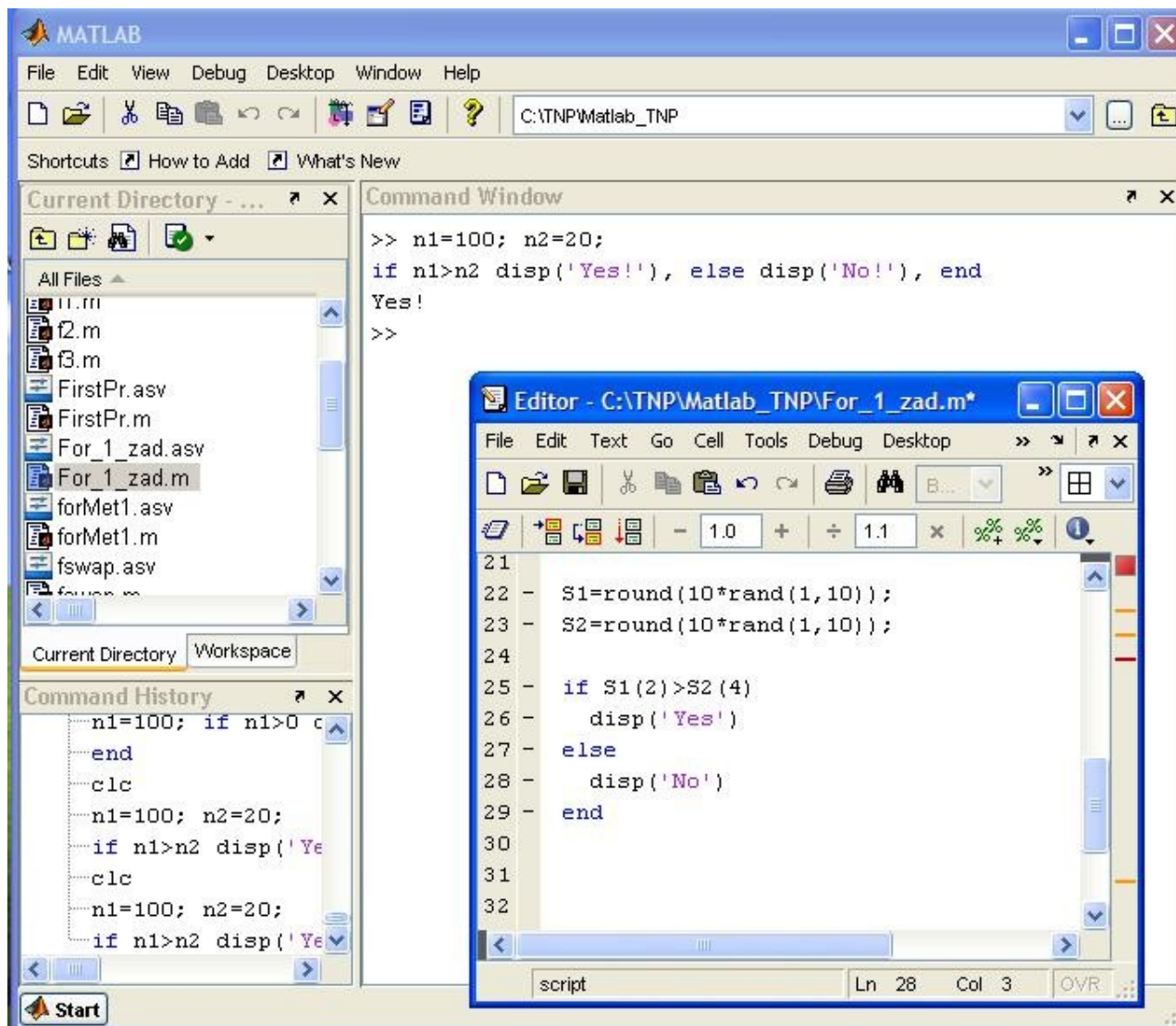
MATLAB (сокращение от MATrix LABoratory) – одна из старейших систем для автоматизации математических расчетов. Это – система программирования. Система применима в любой области науки и техники. Это – расширяемая система, и ее легко приспособить к решению нужных классов задач.

Цель наших занятий – познакомиться с основами программирования на языке этой системы для того, чтобы уметь пользоваться ее возможностями как для демонстрации математических закономерностей при изучении различных разделов математики, так и при проведении собственных исследований на компьютере.

Ниже на рисунке представлено интерфейсное окно системы и окно редактора (Editor), используемое для создания новых программ и функций.

Интерфейсном окне располагаются окна, необходимые для работы:

- главное меню системы для выбора нужной опции;



- окно текущего каталога (Current Directory), имеющее 2 закладки;
- окно с именем Current Directory, раскрывающее содержание выбранного каталога, которое открывается при нажатии на закладку с именем Current Directory;
- окно с именем Workspace, в котором отображаются переменные и их значения, хранящиеся в оперативной памяти;
- окно с именем Command History, в котором отображаются все набранные ранее в окне команд операторы (команды); любую группу этих операторов при желании можно копированием перенести в окно команд;
- командное окно (Command Window), в котором пользователь набирает и исполняет свои команды. Результат работы отображается в этом же окне.

Работать в системе Matlab можно как в командном окне (в режиме прямых вычислений, как на калькуляторе), так и с использованием редактора программ (в режиме программирования). Окно редактора (Editor) также показано на рисунке.

Сначала мы познакомимся с тем, как работать в окне команд. На рисунке Вы видите, что в Command Window написаны (набраны с клавиатуры) следующие операторы (команды):

```
n1=100; n2=20;
```

```
if n1>n2, disp('Yes!'), else disp('No!'), end
Yes!
>>
```

Этот простой пример используем для того, чтобы познакомиться с двумя основными операторами программирования и правилами записи команд в MatLab.

Первый оператор – оператор **присваивания**, в результате выполнения которого две простые переменные с именами n1 и n2 получают значения 100 и 20 соответственно. Оператор присваивания в среде MatLab обозначается обычным знаком равенства, но смысл его совсем другой.

Обратите внимание на то, что в первой строке в качестве **разделителя** команд для присваивания значений переменным n1 и n2 использован символ ;. Кроме указанной функции этот символ применяют также в тех случаях, когда результат работы команд (если это числовые или строковые значения) **не нужно** выводить на экран.

Второй оператор – **условный**. Он предназначен для проверки некоторого условия и определения разных действий при выполнении или не выполнении этого условия. В приведенном примере условие выражается сравнением двух арифметических значений (100 и 20), для чего используется привычный математический знак >. В случае, если значение переменной n1 больше значения переменной n2, на экран должно быть выведено слово Yes!, в противном случае – слово No! Для записи условного оператора используются служебные слова **if, else, end**. Условие и операторы, выполняющиеся в двух разных случаях, также отделяются друг от друга. Как видно, в качестве разделителя может быть применен символ запятой ,.

Третий оператор (функция) – **disp()** используется для вывода на экран результата вычислений. В приведенном примере – это слова 'Yes!' и 'No!'. И обратите внимание, что слова (текстовые значения) должны быть заключены в одиночные апострофы.

Правилами написания команд в окне редактора (Editor) мы займемся позже, а сначала попробуем поработать в командном окне MatLab.

Запомните следующие правила.

1) Каждая команда пишется на отдельной строке после символа >>. Если она не входит на одну строку (80 символов), то может быть продолжена на следующую строку с помощью знака многоточия (три или более точек). Если команды не очень длинные, их можно писать несколько на строке, не забывая использовать разделитель.

```
n1=2; n2=sqrt(n1)
```

2) Для выполнения написанных команд следует нажать клавишу enter. Если все написано верно, и не запрещен вывод на экран, вы увидите результат в следующей форме:

```
n2 =
```

```
1.4142
```

```
>>
```

3) Полученный результат использует по умолчанию короткое (short) представление вещественного числа в фиксированном формате (5 символов). MatLab позволяет выводить результаты вычислений и в других форматах для фиксации большего количества разрядов вещественного результата. Для этого достаточно написать команду **format long**, указав имя формата. Например, для вывода результата вычисления n2 в формате long.

```
format long, n1=2, n2=sqrt(n1) n1 =
```

```
2
```

```
n2 =
```

```
1.41421356237310
```

```
>>
```

4) При наборе команд часто допускаются ошибки. MatLab выдает об этом сообщение с указанием возможной ошибки. Предположим, была сделана ошибка в имени функции, вычисляющей значение квадратного корня (вместо имени `sqr` написано имя `sq`):

```
>> format long, n1=2, n2=sqr(n1)
```

```
n1 =
```

```
2
```

```
??? Undefined command/function 'sqr'.
```

```
>>
```

5) Для того чтобы исправить ошибку, можно снова, уже правильно набрать команду и выполнить ее. Но если команда длинная, удобнее просто выбрать ее из последовательности ранее набранных команд, используя для этого клавиши перемещения курсора: «стрелочка вверх» или «стрелочка вниз», поставив перед этим мигающий курсор следом за знаком `>>`.

6) При записи десятичного числа используется только **точка**.

7) Для получения случайных чисел в MatLab имеется несколько функций. Одной из них является функция **rand**. Без параметров эта функция выдает одно случайное число из интервала (0.0, 1.0). Для того чтобы получать число из большего интервала, например, (0.0, 10.0) или (0.0, 100.0), нужно просто умножить полученное начальное число на соответствующую константу и округлить результат умножения с помощью функции **round()**: `n=round(10*rand)`. Проверьте, как работает команда **rand**. Обратите внимание на то, в каком виде выводится на экран результат.

2. Программирование циклов и условных операторов

MatLab – система, специально предназначенная для проведения сложных вычислений с **векторами, матрицами и многочленами**. По умолчанию система предполагает, что каждая заданная в ней переменная – это вектор или матрица. Все определяется конкретным значением этой переменной.

В примерах первого занятия все вводимые переменные имели по одному единственному значению. То есть, они были векторами, состоящими из одного элемента, значение которого было задано в соответствующем операторе присваивания. Если вектор содержит более одного элемента (массив элементов), то для задания этого вектора в операторе присваивания следует заключить значения элементов в квадратные скобки, отделив значения друг от друга пробелом: `v1=[10 14 8]` или `v2=[3.1415]`. Первый вектор `v1` содержит 3 целочисленных элемента, а второй – 1 элемент, заданный вещественным числом.

Пока ограничимся векторами и на примерах работы с ними рассмотрим еще один важный оператор в программировании – **оператор цикла**.

В MatLab используются циклы **двух** видов: цикл с условием и цикл с известным числом шагов.

Конструкция цикла с условием имеет следующий вид:

```
While условие Тело цикла  
end
```

Тело цикла – это последовательность команд (операторов), которая должна выполняться пока условие истинно. Для записи условий могут применяться следующие операторы отношения (**сравнения**):

| | | | |
|----|--------------|----|------------------------|
| < | – меньше чем | <= | – меньше чем или равно |
| > | – больше чем | >= | – больше чем или равно |
| == | – равно | ~= | – не равно |

Эти операторы применяются для сравнения двух величин или значений, полученных при вычислении арифметических выражений. Они возвращают значения False или True (неверно или верно) в виде чисел 0 или 1 в зависимости от результата сравнения.

Возможно также использование **логических** операций:

&– логическое И (AND)

|– логическое ИЛИ (OR)

~– логическое отрицание (NOT)

Операторы логических операций используются для сравнения условных выражений. Их результатом также являются числа 0 (False) и 1 (True).

Ниже приведен пример использования оператора цикла типа while, для подсчета суммы S значений элементов вектора v1, написанный в командном окне.

```
>> v1=[12 -5 0]; s=0; i=1; while i<4, s=s+v1(i); i=i+1; end
```

В этом цикле переменная с именем i играет роль счетчика числа шагов выполнения цикла. Ее часто называют параметром цикла. Параметр цикла увеличивается на 1 при каждом шаге цикла и входит в его условие. Обратите внимание на выделенные жирным шрифтом операторы, стоящие перед циклом while. Это – операторы,

задающие начальные значения для переменных, участвующих в теле цикла. Цикл должен обязательно заканчиваться служебным словом end.

Второй тип оператора цикла, называемый часто циклом с известным числом шагов, в общем виде имеет следующую конструкцию:

```
for i = StartValue : Different : EndValue
```

```
Тело цикла end
```

Здесь i – параметр цикла, StartValue – его начальное значение, EndValue – конечное значение и Different – приращение параметра i на каждом шаге цикла. Часто используется «укороченная» конструкция:

```
for i = StartValue : EndValue
```

```
Тело цикла end
```

В ходе выполнения такого цикла значение его параметра меняется с шагом +1 от StartValue до EndValue, если начальное значение меньше конечного, и, в противном случае, оно меняется с шагом, равным -1.

Далее приведен пример цикла типа «for», в котором считается сумма s нечетных чисел от 1 до 100.

```
> s=0; for i=1:2:100; s=s+i; end  
> s
```

С помощью цикла типа «for» можно создавать числовые массивы, присваивая каждому элементу массива значение, получаемое с помощью некоторой функции. Массивами в программировании называются наборы однотипных элементов, которые могут быть пронумерованы. Массив обозначается некоторым именем. Номер каждого элемента массива называется его индексом. Обратиться к элементу массива можно, используя его индекс. Это записывается следующим образом: если имя массива, например, `mas`, то `mas (1)` – его первый элемент, `mas (2)` – второй, и т.д.

Индекс может быть **переменной**. Например, `mas (i)` – это *i*-й элемент массива. И если переменной *i* перед обращением к массиву присвоено некоторое значение, например `i=5`, то `mas (i)` покажет значение 5-го элемента массива. При работе с массивами и индексами нужно быть внимательными, чтобы не указывать такое значение для индекса, для которого нет соответствующего элемента. MatLab в этом случае выдаст на экран сообщение об ошибке.

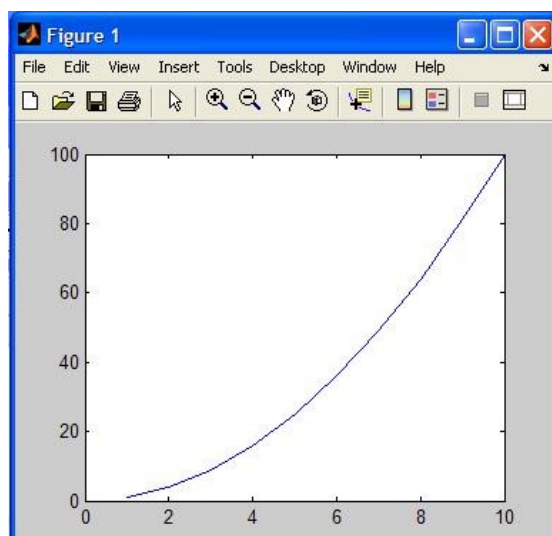
Теперь напишем цикл для получения массива целых чисел. В процессе изучения работы циклических конструкций можно (только для удобства) ограничиться небольшим значением размерности массива (например, 10 элементов). Назовем наш массив, состоящий из 10 квадратов натуральных чисел, – `mas`. Он может быть получен в результате работы следующего цикла.

```
for i=1:1:10; mas(i)=i*i, end
```

Мы можем получить массив случайных значений целых чисел, используя функцию **rand**.

Массивы значений могут быть использованы для построения графиков. Система MatLab имеет достаточно развитые графические средства. Познакомимся с одним из основных операторов графики `plot`. Для того, чтобы построить график некоторой зависимости одной величины от другой, нужно иметь два массива значений – для точек, значения которых откладываются по оси X, и для точек, значения которых откладываются по оси Y. В написанном выше цикле получен массив квадратов чисел от 1 до 10. Назовем его `masY`. Не составляет труда в этом же цикле получить и второй массив. Назовем его `masX`. Это массив самих чисел от 1 до 10. Достаточно после получения массивов написать команду

`plot (masX, masY)` и MatLab выведет на экран параболическую зависимость.



Ниже приведены операторы, написанные в командном окне, для получения графика зависимости значений массива `masY` от значений величины `i`.

- `for i=1:1:10, masX(i)=i, masY(i)=i*i, end`
- `plot(masX, masY)`

3. Набор программ в редакторе и их исполнение

Мы познакомились с тем, как работает система MatLab в том случае, когда команды вводятся в одну строку (она называется командной строкой) в командном окне (Command Window). Это – режим интерпретации и немедленного выполнения команд. Достаточно нажать клавишу «enter» и команда будет выполнена. Вычисленный результат записывается тут же, в командном окне. Однако в системе MatLab есть возможность обработки заранее подготовленной последовательности команд и операторов, записанной в виде файла, который называется **m-файл**. Такая последовательность набирается в окне редактора – **Editor**.

Для того чтобы вызвать окно редактора, в главном меню MatLab выберите опцию File, в выпавшем меню, выберите опцию New, и в следующем появившемся меню выберите опцию M-File. Окно редактора откроется. Можно просто воспользоваться клавишами «ctrl+N».

MatLab m-файлы разделяются на два типа: файлы-сценарии (script-файлы) и процедуры-функции (function-файлы). Эти два типа файлов отличаются друг от друга заголовками, а также их оформлением и использованием.

Script-файл целесообразно начинать со строк с комментариями, поясняющими его назначение. Строки с комментариями начинаются со знака %

```
%Генерация массива целых чисел
%Копирование полученного массива в другой массив for
i=1:1:10,
a(i)=round(10*rand);
end,
bb=aa,
```

Для выполнения последовательности написанных команд в меню редактора **Editor** выберите опцию **Debug** и затем в выпавшем меню опцию **Save and Run**. Если Вы запускаете свою программу первый раз, MatLab потребует от Вас сохранить ее. Имя для файла, в который будет записана последовательность команд, желательно делать содержательным в соответствии с тем, для чего она предназначена, а расширение **.m** MatLab припишет ему сам.

Для того чтобы поупражняться в написании script-программ в редакторе Editor с использованием операторов присваивания, цикла и условного, напишем несколько простых программ, работающих с целыми числами. Для этого нам понадобится несколько функций системы MatLab.

input – функция input обеспечивает ввод данных с клавиатуры, позволяя организовать для этого необходимый диалог.

Например, функция input('Введи число n=') выведет на экран предложение **Введи число n=** с мигающим курсором за знаком равенства, где и следует ввести значение переменной n. Функция выводит его на экран, и далее в программе можно использовать переменную n.

disp – оператор disp обеспечивает вывод значений переменных и текста на экран. Например, оператор disp(x) выведет на экран значение переменной x, а оператор disp('текст – слова') выведет на экран заключенные в апострофы слова. Для того чтобы сделать выводимый результат более наглядным, приходится выводить одновременно и текст, и значения переменных. Например, желательно вывести следующую строку:

```
disp(['Самое большое число равно ', num2str(max)]),
```

Выводимая строка состоит из двух частей – слов, заключенных в апострофы, и преобразованного в строку числового значения переменной с именем max. Преобразование осуществляет функция **num2str(max)**. **Обратите внимание**, что обе эти строки заключаются в квадратные скобки.

fix(x) – функция, возвращающая значение переменной x , округленное до ближайшего целого в направлении к нулю. Например, $\text{fix}(3.14)$ равно 3.

rem(x, y) – функция, возвращающая остаток от деления x на y . **rand** – функция для генерации случайных чисел. Эта функция имеет

много вариантов. Мы пока рассмотрим только 2. Если у функции нет аргументов, она выдает одно случайное число из чисел, распределенных равномерно в интервале (0.0, 1.0). С помощью функции **rand(1, n)** можно получить сразу n чисел, распределенных равномерно в интервале (0.0, 1.0).

round(x) - функция, возвращающая значение, округленное до ближайшего целого.

Ниже приведен пример script-программы для определения, является ли введенное число простым. В программе используется цикл типа `while`. Комментарии помогут понять смысл и назначение команд.

```
% Программа, определяющая является ли введенное число простым
n=input('число n='),      % Ввод числа n
d=2,                    % Выбор первого делителя d
r=rem(n,d),             % Начальное определение остатка r от деления n на d
while r~=0,             % Цикл while до получения нулевого остатка
    d=d+1,               % Тело цикла – увеличение делителя на 1 и
    r=rem(n,d),          % получение нового остатка
end,                     % Конец цикла
if d==n,                % Проверка конечного значения делителя и вывод
    disp([num2str(n), ' - простое число']), else disp([num2str(n), ' -
не простое число']),

end
```



```

...
Цикл while по j                % сравнение i-го элемента с правостоящими
                                ручной способ ввода массива
...
Цикл while по k                % сдвиг влево при равенстве i-го и j-го элем.
...
k=k+1
конец while по k
j=j+1
конец while по j
i=i+1
конец while по i
    
```

Запишем теперь рассмотренный алгоритм на так называемом **псевдокоде** и на русском языке. Учтем, что количество элементов в массиве будет изменяться при удалении каждого повторяющегося элемента. Поэтому при записи алгоритма следует использовать циклы типа while.

Создать массив n целых чисел, в котором есть одинаковые числа; Ввести переменную kol для количества элементов в массиве, Присвоить ей первоначальное значение, равное n;

Ввести переменную-счетчик i для цикла while В цикле while по i от 1 до kol (**i < kol**)

выполнять следующие действия

Ввести переменную-счетчик j для внутреннего цикла while Присвоить j значение, равное i+1 (сосед справа)

В цикле while по j (**j <= kol**)

Все элементы справа от j- ого **по** последний kol **сравнивать** с i-ым элементом

если они равны

Ввести переменную-счетчик k для следующего цикла while Присвоить k значение, равное j (повторяющийся элемент)

В цикле while по k до kol (**k < kol**)

Сдвинуть все элементы массива от (k + 1)-го до kol-го на одну позицию влево

увеличить счетчик k на 1 конец цикла while по k

уменьшить kol на 1

иначе перейти к следующему значению j (j+1) конец цикла while по j

перейти к следующему значению i (i+1) конец цикла while по

i

%Генерация массива целых чисел с циклом и без цикла

n=input('vvedi kolichество elem n='),

*% for i=1:1:n, a(i)=round(10*rand); end,*

a=round(10*rand(1, n)); n=input('vvedi kolich elem n='),

a=[], an=[],

a=[12 4 67 0 23 12 45 10 0 55]

*%a=round(10*rand(1, n)); %in MatLab*

```

% Удаление повторяющихся элементов из массива
i=1; kol=n;           %инициализация переменных
while i<=kol-1;      %Просмотр элементов от первого до предпоследнего
j=i+1;              %Выбираем соседний с (i-ым) элемент (j-й)
-   просматриваем в цикле все элементы справа от j-ого до последнего
1.   Сравниваем каждый из элементов с (i-ым)
-   Если элементы равны, нужно сдвинуть влево все элементы от (j-ого)
-   до последнего и уменьшить на 1 количество элементов в массиве
while j<=kol;
if a(i)==a(j); k=j;
while k<kol; % цикл для сдвига a(k)=a(k+1); % поэтому k<kol, а не k<=kol
k=k+1;
end
kol=kol-1; % Уменьшаем на 1 число элементов в массиве else j=j+1; %если элем. не равны
друг другу, переходим к след.
end, end,
i=i+1;          % Выбираем новый элемент в массиве для сравнения с правыми
end

% цикл для копирования оставшихся после удаления элементов в новый массив
for in=1:1:kol; an(in)=a(in);
end
    
```

5. Функции и работа с ними

Центральным моментом программирования в среде MatLab является задание новых функций. Для функций, вычисляющих одно значение, используется следующая конструкция:

```

function имя_переменной = имя_функции (список входных параметров)
% Комментарии, указывающие назначение функции Операторы
%комментарии операторов
...
имя_переменной = выражение           %Возвращаемое значение
end
    
```

имя функции – это имя файла, в котором будет сохранен текст программы функции. Прежде чем сохранять функцию с некоторым именем, следует проверить, не использовалось ли такое имя в среде MatLab для сохранения собственных функций.

При написании программы объединения множеств, мы дважды использовали одинаковый фрагмент операторов для удаления повторяющихся элементов в массиве значений. Попробуем сейчас оформить этот фрагмент как функцию с одним входным параметром – это массив, в котором имеются повторяющиеся элементы, и одним результирующим массивом, в котором нет повторяющихся значений.

При сохранении m-файла, содержащего текст программы функции, назовем его delrepel (сокращение от английских слов **delete**, **repeat**, **element**) и возвращаемое значение функции movmas.

Входным параметром функции и возвращаемым значением являются в нашем случае **массивы - mas** и **movmas**.

```
function movmas = delrepele(mas)
```

%Удаляет повтор-ся элементы из массива mas (входной параметр) %Возвращает массив movmas, в котором повторения элементов нет movmas=[]; %очищаем выходной массив

nf=length(mas); %Определяем длину входного массива masf=[]; %Задаем локальный массив для тела функции for i=1:nf, %Копируем в него значения входного массива

```
masf(i)=mas(i);
```

```
end,
```

```
i=1;
```

```
while i<=nf-1;
```

```
    j=i+1; while j<=nf;
```

```
        if masf(i)==masf(j) k=j;
```

```
            while k<nf; %Сдвигаем элементы налево, удаляя %совпадающий
```

```
            masf(k)=masf(k+1);
```

```
            k=k+1;
```

```
            end;
```

```
            nf=nf-1; %Уменьшаем на 1 число элементов в локальном %массиве
```

```
            else j=j+1; %Переходим к следующему элементу для %сравнения
```

```
            end; %Конец оператора if end; %конец цикла по j
```

```
            i=i+1;
```

```
            end; %конец цикла по i
```

```
        for i=1:nf;
```

```
            movmas(i)=masf(i); %Копируем локальный массив в выходной
```

```
        end end
```

Воспользуемся теперь этой функцией для написания программы получения пересечения двух множеств.

%Пересечение множеств anov и bnov и запись результата в массив

%cOut

```
a=[]; anov=[]; b=[]; bnov=[]; cOut=[];
```

%Очистка массивов

```
a=[3 6 8 8 1 9 12 6] %Задание первого массива
```

```
b=[5 6 7 9 12]
```

%Задание второго массива

```
anov = delrepele(a)
```

%Удаление повторяющихся в первом массиве

```
kola=length(anov)
```

```
bnov = delrepele(b)
```

%Удаление повторяющихся во втором массиве

```
kolb=length(bnov)
```

```
ki=1;
```

&Счетчик для числа элементов в массиве-пересечении

%Запись алгоритма на псевдокоде

Цикл по i для перебора всех элементов массива $anov\ j=1; priznak=1;$
 цикл по j для нахождения равных $anov(i)$ и $bnov(j)$
 если $anov(i) == bnov(j)$

записываем любой из них в выходной массив увеличиваем счетчик в массиве-пересече-
 нии присваиваем переменной `priznak` ноль

иначе увеличиваем счетчик массива `bnov` на 1 конец условного оператора

конец цикла по j конец цикла по i

опред. длину выходного массива с помощью Функции `length` Выводим выходной массив

6. Программирование алгоритмов сортировки

Написать функцию, которая осуществляет поиск максимального элемента в массиве целых чисел и его позиции (индекса) в этом массиве.

Рассмотрим пример: пусть имеется массив `inMassiv` с заданным числом элементов: 5 -8 12 33 7 4. Максимальный элемент в нем имеет значение 33, а его позиция в массиве – 4 (он стоит на 4-ом месте).

Создаваемая функция должна возвращать два значения – значение величины максимального элемента (`valmax`) и значение его позиции (`posmax`). Следовательно, выходным значением этой функции будет массив значений. Назовем его `outMassiv`. Он будет содержать всего два значения – на первом месте будет записана величина максимума, на втором – его позиция.

Таким образом, заголовок нашей функции, которую мы назовем `myMax` (т.е. с таким именем мы ее сохраним), может выглядеть так:

`function outMassiv = myMax(inMassiv).`

```
% Поиск максимального элемента в массиве и его позиции
function outMassiv = myMax(inMassiv)
    nf=length(inMassiv),           %определение длины входного массива
    outMassiv = [],
    valmax =           inMassiv(1), %Начальное присваивание для valmax
    posmax =           1;          %Начальное присваивание для
for i = 2:nf                 %Цикл для поиска очередных значений valmax и posmax
    if valmax < inMassiv(i),
        valmax = inMassiv(i),
        posmax = i,
    end
end
outMassiv = [valmax posmax], %Получение выходного массива
end
```

Пример обращения к функции и вывода результата

```
>> aInp=[3 -9 0 12 44 33 6],      aOut=mymax(aInp),
disp(['максимум = ',num2str(aOut(1))]),
disp(['позиция = ',num2str(aOut(2))])
```

```
aInp = 3      -9      0      12      44      33      6
```

aOut = 44 5

максимум = 44 позиция = 5

Алгоритм упорядочения (сортировки) элементов массива по возрастанию с использованием поиска максимального элемента массива

Этот алгоритм представляет собой следующую последовательность шагов: находим в массиве максимальный элемент и переставляем его с последним элементом, иными словами, ставим максимальный элемент на правильное место. Затем уменьшаем число просматриваемых на следующем шаге элементов на 1, т.к. один уже стоит на месте. Опять находим максимальный элемент, но уже в сокращенном массиве, и переставляем его с последним в этом сокращенном массиве. Уже два элемента будут стоять на месте. Снова уменьшаем число просматриваемых на следующем шаге элементов на 1. И так далее, пока не упорядочим весь массив. Ниже приведен пример перестановок для массива из 5 элементов: 13 5 12 2 4 На первом шаге меняются местами элементы 13 и 4:

| | | | | | |
|---------------------|---|---|----|----|----|
| | 4 | 5 | 12 | 2 | 13 |
| На втором: 12 и 2 | 4 | 5 | 2 | 12 | 13 |
| На третьем: 5 и 2 | 4 | 2 | 5 | 12 | 13 |
| На четвертом: 4 и 2 | 2 | 4 | 5 | 12 | 13 |

После 4-х шагов массив получится упорядоченным.

Можете проделать эту процедуру с массивами с другим количеством элементов, и вы убедитесь, что число повторений процедуры всегда будет на 1 меньше количества элементов в массиве.

Ниже приведен алгоритм сортировки на псевдокоде. Перепишите его на языке Matlab. Учтите, что в приведенном псевдокоде `sorting` - имя файла, в котором должна быть сохранена функция.

```
function sortmassiv = sorting(inmassiv)
```

Присвоить переменной `nf` значение числа элементов во входном массиве Создать локальный массив `locmassiv` с таким же числом элементов Запомнить место последнего элемента массива в переменной `lastplace` Оформить внешний цикл с нужным числом повторений (`nf-1`)

Сделать начальные присваивания значениям максимума `maxx` и его места

...

цикл для поиска максимального элемента `maxx` и его места `maxplace`

...

end

Поменять местами максимальный элемент и последний уменьшить номер последнего элемента в массиве на 1

end

Скопировать локальный массив в выходную переменную `sortmassiv`

end

7. Символьные данные и текстовые строки

Во всех языках программирования большую роль играют обработка и хранение текстовых данных (текстов на естественных языках). В МатЛаб для этой цели предусмотрено несколько возможностей. Прежде всего, это – специальный **символьный** тип данных `char` (сокращение от английского слова `character` – символ, знак, буква, литера).

Каждому возможному символу (букве алфавита или специальному символу) в соответствии со стандартными таблицами кодировок ставится в соответствие целое числовое значение, для хранения которого в памяти машины всегда достаточно 2 байтов памяти. Именно 2 байта и отводятся для хранения каждого элемента символьного массива в системе МатЛаб.

Задать элемент символьного массива можно двумя способами.

1. Целым числовым кодом, к которому применим модификатор char: `sim1(1)= char(97);`

Здесь задан символьный массив `sim1`, с единственным элементом, равным английской букве `a`. Согласно всем стандартизованным на сегодняшний день кодовым таблицам (так называемый код ASCII) числовой код английской буквы `a` равен 97. Вы можете убедиться в этом, если в командном окне напишете и выполните команду `whos sim1`. Результат приведен ниже:

```
whos sim1
Name          Size      Bytes      Class
sim1          1*1        2          char array
Grand total is 1
sim1
ans = a
```

2. Во-вторых, того же результата можно добиться применением *апострофов*: `sim1(1) = 'a';`

При этом способе не требуется помнить числовой код английской буквы `a`. Чтобы узнать код того или иного символа, нужно применить к этому символу модификатор `double`:

```
double('b') ans =
98
```

Первые 128 символов во всех таблицах кодировок совпадают (это и есть стандартная кодировка ASCII). Коды от 0 до 31 соответствуют так называемым управляющим символам, не имеющим визуального представления.

Символы для числовых кодов, больших 127, не определены столь же однозначно и зависят от конкретных вариантов кодировок. *Русскому языку* в операционной системе Windows соответствует кодировка 1251. В ней присутствуют символы кириллицы.

Символы для кодов от 32 и далее (примерно до 1170) можно увидеть в командном окне системы МатЛаб с помощью команд, подобных написанной ниже.

```
>> char(32:47)
ans =
!"#$%&'()*+,-./
```

Обратите внимание, что в ответе вы видите 15 символов, а не 16 (от 32 до 47). Это потому, что первым символом ответа является символ пробела («пустое» место перед восклицательным знаком) с кодом 32. Убедитесь в этом, выполнив в командном окне следующую команду:

```
>> double(' ')
```

ans =

32

Создание символьных массивов

Символьные массивы из нескольких символов создаются как обычными для всех массивов системы МатЛаб операциями **конкатенации** и **индексации**, так и специальной символьной операцией конструирования массива, когда все символы массива записываются подряд в квадратных скобках, причем каждый символ ограничивается с двух сторон апострофами.

Ниже приведены различные варианты создания символьных одномерных массивов:

```
mas1 = ['a', 'b', 'c', 'd'];
```

```
mas1(1) = 'a'; mas1(2) = 'b'; mas1(3) = 'c'; mas1(4) = 'd'; mas2='Hello, World!';
```

1. результате таких присваиваний создаются переменные mas1 и mas2
 - символьные массивы типа char.

Так как под каждый символ отводится 2 байта памяти, то переменная mas1 занимает в памяти компьютера 8 байт, поскольку этот массив имеет

тип char и всего содержит 4 элемента. Массив mas2 содержит 13 символьных элементов, включая символ запятой и символ пробела, и занимает в памяти 26 байт.

Модификатор char можно применять сразу к нескольким числовым элементам. Например, операторы присваивания

```
x = [97, 98, 99, 100]; mas1 = char(x)
```

дают то же самое символьное значение ('abcd') для переменной mas1, которое было получено выше другим способом.

```
>> mas1 = ['a','b','c','d'], mas2=char(mas1), mas3=double(mas2),
```

```
mas1 =      abcd
```

```
mas2 =      abcd
```

```
mas3 =      97          98          99  100
```

```
>> mas4 = char(mas3)
```

```
mas4 =      abcd
```

Напишите алгоритм шифрования текста по следующим правилам: После каждой гласной буквы вставлять строку из двух букв, состоящую из буквы "к" и этой гласной буквы.

Алгоритм шифровки состоит в следующем:

- 1) Нужно просмотреть весь введенный текст (строку s) от 1-го символа до n-го (это – цикл);

- 2) На каждом шаге цикла проверять, не является ли очередной символ гласной буквой (для этого оформить отдельную функцию `glasnaia`);
- 3) В зависимости от результата проверки по-разному формировать зашифрованный текст – строка `snew`:

Если *i*-я буква является гласной, то после нее вставляются модифицирующие буквы,

иначе – в строку `snew` просто добавляется очередной символ из строки `s`.

Алгоритм дешифровки состоит в следующем:

- 1) Нужно просмотреть весь зашифрованный текст (`sNew`) от 1-го символа до конечного, равного длине строки, получившейся после шифрования.
- 2) На каждом шаге цикла проверять, не является ли очередной символ гласной буквой (для этого используется функция `glasnaia`);
- 3) В зависимости от результата проверки по-разному формировать дешифрованный текст (`sResult`): если выделенная буква в строке `sNew` является гласной, она добавляется в строку `sResult` и счетчик цикла увеличивается на 3 (пропускаются добавленные при шифровке буквы); иначе – буква добавляется, и счетчик цикла увеличивается только на 1.

Второй способ шифрования текста

Он описан в книге М. Зуева-Ордынца «Сказание о граде Ново-Китеже». Для переписки царского двора с русскими посланцами, находившимися за границей, использовался следующий способ шифровки: все согласные буквы русской азбуки записывались в два ряда; одна половина букв – вверху, другая половина – внизу, причем в обратном порядке (одна буква под другой).

б в г д ж з к л м н щ ш ч ц х ф т с р п

При зашифровке слов согласные взаимно заменялись, а гласные буквы и буквы **й, ъ, ь** вписывались без изменений. Слова записывались без промежутков между ними, как вообще писался любой текст до 16 века, и это еще более затрудняло разгадывание текста.

Функции MatLab для обработки символьных строк

Общие (General)

1. `char`– сформировать массив символов (строку)
2. `double` – преобразовать символы строки в числовые коды
3. `cellstr`– преобразовать массив символов в массив ячеек для строк
4. `blanks`– сформировать строку символов
5. `deblank` – удалить пробелы
6. `eval`– выполнить выражение, записанное в виде строки символов

Проверка строк (String tests)

- `ischar`– истинно, если это массив символов
- `iscellstr`– истинно, если это массив ячеек для строк
- `isletter` – истинно, если это символ (буква) алфавита
- `isspace` – истинно, если это пробел

Операции над строками

A = [3 1 -2; 5 8 4].

- strcat– горизонтальное объединение строк
- strvcat– вертикальное объединение строк
- strcmp– сравнить строки
- strncmp – сравнить первые n символов строк
- strcmpi – сравнить строки, игнорируя регистр
- strncmpi – сравнить n символов строк, игнорируя регистр
- findstr– найти заданную строку в другой строке
- strjust– выровнять массив символов
- strmatch – найти все совпадения
- strep– заменить одну строку другой
- strtok– найти часть строки, ограниченную разделителями (token)
- upper– перевести все символы строки в верхний регистр
- lower– перевести все символы строки в нижний регистр

Преобразования строк (String to number conversion)

- num2str– преобразование числа в строку
- int2str– преобразование целого числа в строку
- mat2str– преобразование матрицы в строку
- str2double – преобразование строки в число с удвоенной точностью
- str2num– преобразование массива строк в числовой массив
- sprintf– записать форматированное значение в виде строки
- sscanf– прочитать строку с учетом формата

8. Двумерные массивы (матрицы)

Ввод матриц и работа с элементами матриц

Небольшие по размеру матрицы удобно вводить прямо из командного окна. Для этого используются, как и в случае одномерного массива, квадратные скобки. Последовательности числовых значений строк отделяются друг от друга символом точки с запятой.

Напишите и выполните следующую команду: Вы получите ответ:

A =

```
3    1    2
5    8
```

Другой способ ввода - построчно. Начните с квадратной скобки. Вводите отдельно каждую строку, нажимая enter после завершения строки. После ввода последней строки поставьте закрывающую квадратную скобку.

```
В      -  
= [3   2      ;нажмите enter  
      4  
      5   ] ;поставьте закрывающую скобку
```

Вы получите ответ:

В =

```
3   1   2  
5   8
```

Еще один способ ввода заключается в том, что матрицу можно трактовать как вектор строку (одномерный массив), каждый элемент которой является вектором-столбцом. Например, нашу матрицу можно ввести так:

```
C=[[3; 5] [1; 8] [-2; 4]]
```

Если во внутренних квадратных скобках убрать символ ;, получится один одномерный массив из 6 элементов, полученный сцеплением (конкатенацией) трех одномерных массивов, каждый из которых состоит из двух элементов:

```
C=[[3 5] [1 8] [-2 4]]
```

Обращение к элементам матриц осуществляется при помощи двух индексов строки и столбца, заключенных в круглые скобки:

```
t = C(2, 3) t = 4
```

9. Массивы ячеек

До сих пор мы работали с линейными (или одномерными) массивами. МатЛаб – **матричная** среда программирования. Основным типом данных в этой среде является именно **матрица** (таблица элементов одного типа).

Двумерный массив (матрицу) можно рассматривать как **массив массивов** (массив строк или массив столбцов).

```
>> numMas=[1 2 3 4; 5 6 7 8; 9 4 3 1]
```

В обращении к элементу массива всегда используется 2 индекса (номер строки и номер столбца):

```
>> numMas(2, 3)
```

```
ans = 7
```

Рассмотрим использование двумерных массивов для представления текстовой (строковой) информации. Начнем с примеров.

При выполнении (в окне команд) нижеследующей команды будет создана переменная с именем names1 и ей будет присвоено значение.

```
>> names1='Василий'  
  
names1 = Василий
```

При выполнении (в окне команд) команды `whos names1` будут выведены на экран атрибуты этой переменной: имя, размер, число используемых байтов и имя класса, к которому относится переменная.

```
>> whos names1  
Name                Size          Bytes          Class  
names1              1x7           14             char array  
  
Grand total is 7 elements using 14 bytes
```

Создадим вторую переменную с именем `names2` как **массив** нескольких имен:

```
>> names2 = ['Василий','Павел','Марина']  
  
names2 = ВасилийПавелМарина
```

Однако команда `whos names2` нам покажет, что в результате получается не массив, состоящий из трех имен (массив массивов), а один символьный массив, в котором все три имени сцеплены вместе.

```
>> whos names2  
Name                Size          Bytes          Class  
names2              1x18          36             char array  
  
Grand total is 18 elements using 36 bytes
```

Воспользуемся функцией `char` для создания массива имен:

```
>> names3 = char('Василий','Павел','Марина')
```

```
Names3 =
```

```
Василий  
Павел  
Марина
```

```
>> whos names3  
Name                Size          Bytes          Class  
Names3              3x7           42             char array  
  
Grand total is 21 elements using 42 bytes
```

Проанализируйте результат выполнения команды `whos names3`. Атрибут `Size` говорит о том, что создана переменная, являющаяся

таблицей, состоящей из 3 (трех) строк, каждая из которых состоит из 7 столбиков-букв. Количество столбцов в матрице при создании переменной `names3` функция `char` определила как количество букв в самом длинном имени ('Василий').

Недостающие столбцы в более коротких именах эта функция заполнила символом пробела. Обратите также внимание на то, что эта функция создает вектор-столбец в отличие от предыдущих способов.

Рассмотренный способ формирования текстовых данных (с лишними пробелами) вряд ли является удобным при написании алгоритмов их обработки. Поэтому мы рассмотрим новый тип данных системы МатЛаб, который носит название **cells (ячейки)**.

Тип данных `cells` (ячейки) является агрегированным типом данных в системе МатЛаб. **Массив ячеек (cell array) содержит в качестве своих элементов массивы разных типов.** Поэтому можно сказать, что массив ячеек является массивом массивов. По сути, он является **универсальным контейнером** – его элементы (ячейки) могут содержать любые типы данных, с которыми работает МатЛаб. Подчеркнем, что важно различать ячейку (элемент массива ячеек) и ее содержимое.

В системе МатЛаб выражение любого типа можно превратить в ячейку, заключив его в **фигурные скобки**. Таким образом, фигурные скобки являются **конструктором** ячеек аналогично тому, как квадратные скобки являются конструктором числовых массивов, а апострофы – конструктором символьных массивов.

Чтобы подобраться к **содержимому** ячеек, нужно **индексировать массив ячеек при помощи фигурных скобок**. При обычной индексации круглыми скобками мы из массива ячеек извлекаем отдельную ячейку, которая сама является массивом.

Массивы ячеек полностью решают типовую задачу хранения нескольких строковых данных под одним именем.

Раньше мы формировали матрицы типа `char`, каждая строка которых **обязана** была иметь одну и ту же длину. Это очевидным образом ограничивает применение такого решения. В случае массива ячеек такого ограничения нет. Массив ячеек, содержащий в качестве своих элементов текстовые строки, часто называют просто **массивом строк**.

Пустые ячейки обозначаются `{[]}`. Затем можно проводить поэлементные присваивания. Многие из функций обработки строк могут принимать в качестве своих параметров массивы строк (массивы ячеек, содержащих строки).

Применение массива ячеек для хранения набора строк удобнее, так как **не требует** искусственного выравнивания длин строк концевыми пробелами.

Задать слова

```
>> str1 = 'идет', str2 = 'волшебница', str3 = 'зима'
```

Создать массив слов

```
>> inpar=[str1 str2 str3]
```

```
inpar = идетволшебницазима
```

Посмотреть атрибуты

```
>> whos inpar
```

| Name | Size | Bytes | Class |
|------|------|-------|-------|
|------|------|-------|-------|

```
inparr      1x18      36      char array
```

Grand total is 18 elements using 36 bytes

Создать массив слов с помощью функции char

```
>> inpchar=char(str1, str2, str3)
inpchar =
```

```
идет
волшебница
зима
```

Посмотреть атрибуты

```
>> whos inpchar
```

| Name | Size | Bytes | Class |
|---------|------|-------|------------|
| inpchar | 3x10 | 60 | char array |

Grand total is 30 elements using 60 bytes

Создать массив ячеек со словами

```
>> inpcell={str1, str2, str3} inpcell
```

=

```
'идет' 'волшебница'      'зима'
```

Посмотреть атрибуты

```
>> whos inpcell
```

| Name | Size | Bytes | Class |
|---------|------|-------|------------|
| inpcell | 1x3 | 216 | cell array |

Grand total is 21 elements using 216 bytes

Транспонировать матрицу

```
>> inpcelln=inpcell' in-
```

```
pcelln =
```

```
'идет' 'волшебница'
'зима'
```

Посмотреть атрибуты

```
>> whos inpcelln
```

| Name | Size | Bytes | Class |
|----------|------|-------|------------|
| inpcelln | 3x1 | 216 | cell array |

Grand total is 21 elements using 216 bytes

10. Структуры и массивы структур

Рассмотренные ранее обычные массивы удобны при работе с однородными данными – только числами или только текстовыми строками. Однако информация при решении большого числа задач может быть представлена как набор элементов разного типа. В качестве простого примера рассмотрим информацию о группе студентов, включающую в себя следующие данные **для каждого** студента – фамилию и имя студента, год его рождения, результаты сдачи экзаменов. В языке MatLab для представления подобных данных имеется специальный тип, называемый **структурой**.

Для создания структуры используется команда (функция), имеющая следующий формат:

Имя_структуры = **struct**(имя_поля1, значение1, имя_поля2, значение2,...), где

именами полей являются строки или строковые переменные, а значениями – данные любых типов, включая числовые массивы, строки, массивы строк и структуры. Для нашего примера команда, описывающая структуру для каждого студента, имеет вид:

```
student = struct('Family', 'ooo', 'Name', 'oo', 'Year', '0000', 'Marks', [0 0 0 0])
```

Наберите в командном окне написанную выше команду и выполните ее. Вы сразу увидите структуру со всеми ее полями и их значениями (у нас они пока формальные – нулевые). Это пока только **вид** структуры.

```
student =
```

```
Family: 'ooo'
```

```
Nome: 'oo' year: '0000' Marks: [0 0 0 0]
```

Все поля нашей структуры можно также увидеть, воспользовавшись встроенной функцией **fieldnames**:

```
>> names = fieldnames(student)
```

```
names = 'Family' 'Nome'
```

```
'Year' 'Marks'
```

Если в окне команд выполнить команду

```
>> whos
```

| names | Name | Size | Bytes | Class |
|-------|-------|------|-------|------------|
| names | names | 4x1 | 278 | cell array |

Grand total is 23 elements using 278 bytes, можно увидеть, что все имена полей структуры представляют собой **массив ячеек**. Для **выделения содержимого** ячейки из массива следует указать ее номер (индекс) в **фигурных скобках** после имени массива ячеек.

Обращение к полю структуры осуществляется с помощью **составного имени**, которое представляет собой имя структуры и имя поля, разделенные **символом точки**.

```
>> student.Marks
```

```
ans =  
0      0      0      0
```

Можно для получения того же результата воспользоваться полученным ранее массивом ячеек с именами полей:

```
;Определяем имя 4-ого элемента структуры через массив ячеек  
>> fn=names{4}
```

```
fn = Marks  
>> ss = student.(fn)      ;Обратите внимание на круглые скобки!
```

```
ss =  
0      0      0      0
```

Массивы структур

Проиллюстрируем работу с массивами структур на примере информации о группе студентов, которая представлена в нижеследующей таблице.

| № | Фамилия | Имя | Год рождения | Оценки по предметам | | | |
|---|---------|--------|--------------|---------------------|----|-----|----|
| | | | | I | II | III | VI |
| 1 | Акулова | Нина | 1995 | 5 | 5 | 4 | 4 |
| 2 | Кетов | Семен | 1996 | 3 | 4 | 5 | 3 |
| 3 | Окунев | Иван | 1995 | 3 | 3 | 4 | 4 |
| 4 | Рыбкин | Андрей | 1994 | 5 | 5 | 5 | 5 |
| 5 | Щукина | Ирина | 1996 | 4 | 4 | 4 | 4 |

Для хранения такой информации естественно использовать **массив структур**, каждый элемент которого является **структурой** с одинаковым набором полей (для примера возьмем описанную ранее структуру student).

Назовем этот массив структур Group. Как следует из таблицы, массив структур в нашем примере будет состоять из 5 элементов.

Допускаются **два способа** создания и заполнения массива структур: - операторами **присваивания** для всех полей **каждой** структуры;

- функцией `struct`, позволяющей занести значения сразу **во все** поля структуры.

Добавление нового поля во все структуры массива производится с помощью оператора присваивания, в левой части которого задается название нового поля, а в правой части – его значение для соответствующей структуры массива.

В конце файл-программы, в которой создается массив структур `group`, напишите следующий оператор присваивания: `group(1).ball=18`; (18 – сумма баллов для первого студента). Выполните программу и вы увидите, что поле **ball** появилось во всех структурах, но оно является пустым.

Цель третьего задания написать команду (операторы цикла), вычисляющие суммарный балл для каждого студента и помещающие его в новое поле каждой структуры. Ниже расположены пояснения для написания этой команды:

Создать новое поле для первой структуры массива структур; Определить длину массива структур, если она не определялась ранее; В цикле от 2 до длины массива структур вычислить сумму баллов для каждого студента:

```
sum=0;
```

цикл для вычисления суммы (`sum`) баллов, находящихся в поле `marks` каждой структуры;

```
end цикла для sum;
```

Записать значения сумм в соответствующие поля структур; `end` цикла для массива структур

В массиве структур у каждого элемента можно **удалить поле**, если оно стало не нужно. Так же как добавление нового поля делается одной командой для всех элементов массива структур, удаление поля делается тоже одной командой для всех элементов массива структур. Первым входным аргументом этой команды является имя массива, а вторым – строка или строковая переменная с именем удаляемого поля. Преобразованная структура возвращается в выходном аргументе команды с именем `rmfield`.

Если нужно, например, удалить поле с именем `year` из созданного нами массива структур следует написать такую команду:

```
groupnew = rmfield (group, 'year');
```

`groupnew` –имя нового массива структур с удаленным полем.

%Файл-программа заполняет массив структур в соответствии с

```
%данными таблицы group = struct('Family', 'ooo', 'Name', 'oo', '%year', 0000, 'Marks', [0 0 0  
0])
```

```
group(1)= struct('Family', 'Акулова', 'Name', 'Нина', 'year', 1995, 'Marks', [5 5 4 4]);
```

```
group(2)= struct('Family', 'Кетов', 'Name', 'Семен', 'year', 1996, 'Marks', [3 3 5 3]);
```

```
group(3)= struct('Family', 'Окунев', 'Name', 'Иван', 'year', 1995, 'Marks', [3 3 4 2]);
```

```
group(4)= struct('Family', 'Рыбкин', 'Name', 'Андрей', 'year', 1994, 'Marks', [5 5 5 5]);
```

```
group(5)= struct('Family', 'Щукина', 'Name', 'Ирина', 'year', 1996, 'Marks', [2 4 4 4]);
```

```
group(6)= struct('Family', 'Карасев', 'Name', 'Сергей', 'year', 1995, 'Marks', [3 3 4 3]);
```

```
group(7)= struct('Family', 'Семгин', 'Name', 'Федор', 'year', 1995, 'Marks', [3 3 3 3]);
```

```
for i=1:7
    disp([group(i).Family, ', ', group(i).Name, ', ', num2str(group(i).year)])
end len=length(group) for
j=1:len
    disp (group(j))
end
```

Второй вариант заполнения массива структур

```
%Заполнение первой структуры массива group(1) group(1).Family = 'Акулова';
group(1).Name = 'Нина'; group(1).year= 1995; group(1).Marks= [5 5 4 4];
%Заполнение второй структуры массива group(2) group(2).Family = 'Кетов';
group(2).Name = 'Семен'; group(2).year= 1996; group(2).Marks= [3 3 5 3];
%Заполнение третьей структуры массива group(3) group(3).Family = 'Окунев';
group(3).Name = 'Иван'; group(3).year= 1995; group(3).Marks= [3 3 4 2];
%Заполнение четвертой структуры массива group(4) group(4).Family = 'Рыбкин';
group(4).Name = 'Андрей'; group(4).year= 1994; group(4).Marks= [5 5 5 5];
%Заполнение пятой структуры массива group(5) group(5).Family = 'Щукина';
group(5).Name = 'Ирина'; group(5).year= 1996; group(5).Marks= [2 4 4 4];
%Заполнение шестой структуры массива group(6) group(6).Family = 'Карасев';
group(6).Name = 'Сергей'; group(6).year= 1995; group(6).Marks= [3 3 4 3];
%Заполнение первой структуры массива group(7) group(7).Family = 'Семгин';
group(7).Name = 'Федор'; group(7).year= 1995; group(7).Marks= [3 3 3 3];
```

```
len=length(group) for j=1:len
    disp (group(j))
end
```

```
%Создание нового поля ball group(1).ball=18; len=length(group);
For j=2:len b=0,
for k = 1:length(group(1).Marks), b=b+group(j).Marks(k),
end group(j).ball=b;
end
```

```
%Добавление нового поля meanball в структуру
group(1).meanball=26 / 6;
For j=2:len mb=group(j).ball / 6;
group(j).meanball=mb;
end
```

```
%Вывод массива структур на экран (показано ниже конца программы) for j=1:len
disp (group(j))
end
```

```
Family: 'Акулова' Name: 'Нина' year: 1995
```

Marks: [5 5 4 4 3 5] ball: 26
meanball: 4.3333

Family: 'Кетов'
Name: 'Семен' year: 1996
Marks: [3 3 5 3 4 4] ball: 22
meanball: 3.6667

Family: 'Окунев'
Name: 'Иван' year: 1995
Marks: [3 3 4 2 4 4] ball: 20
meanball: 3.3333

Family: 'Рыбкин'
Name: 'Андрей' year: 1994
Marks: [5 5 5 5 5 5] ball: 30
meanball: 5

Family: 'Щукина'
Name: 'Ирина' year: 1996
Marks: [2 4 4 4 3 3] ball: 20
meanball: 3.3333

Family: 'Карасев'
Name: 'Сергей' year: 1995
Marks: [3 3 4 3 4 5] ball: 22
meanball: 3.6667

Family: 'Семгин'
Name: 'Федор' year: 1995
Marks: [3 3 3 3 3 3] ball: 18
meanball: 3

11. Работа с файлами

Наличие большого количества информации требует организации **работы с файлами**, в которые информация может записываться, изменяться там, в случае необходимости, и читаться из них для обработки соответствующими программами.

Программирование работы с файлами включает в себя следующие этапы: открытие файла, чтение или запись информации в файл и закрытие файла. Файлы можно использовать только для чтения, только для записи и для чтения и записи одновременно.

Для **открытия существующего или создания нового файла** применяется команда **foren** ('полное_имя_файла', 'параметр').

Параметр может быть одним из следующих:

'rt' – открываемый текстовый файл предназначен только для чтения; 'rt+' – открываемый текстовый файл предназначен для чтения и записи; 'wt' – создаваемый пустой текстовый файл предназначен только для записи;

'wt+' – создаваемый пустой текстовый файл предназначен и для чтения и для записи;

'at' – открываемый текстовый файл предназначен только для добавления данных в конец файла (если файл не существует, то он создается);

'at+' – открываемый текстовый файл предназначен только для добавления данных в конец файла и чтения данных (если файл не существует, то он создается).

Если файл открыть не удалось, команда выдает значение, равное **-1**. Ошибки чаще всего возникают, если МатЛаб не может найти

требуемый файл. Поэтому лучше всего задавать в команде полное имя файла.

Чтение строк из открытого текстового файла производится командой **fgetl('идентификатор файла', 'строковая переменная')**.

Каждое обращение к этой команде позволяет последовательно считывать строки по одной от начала файла до его конца.

Достижение конца файла фиксируется функцией **feof** с входным аргументом – идентификатором файла. Функция **feof** возвращает **1**, если в файле закончились все строки, и **0** в противном случае.

По окончании работы с файлом его следует закрыть командой **fclose('идентификатор файла')**.

Ниже приведен текст **функции**, позволяющей прочитать текст из файла и отобразить его в командном окне.

Создайте несколько текстовых файлов и сохраните их в папке, в которой вы сохраняете m-файлы. Наберите эту функцию и посмотрите, как она работает.

```
function mas = myview(filename)
```

%Функция выводит содержимое текстового файла на экран %и сохраняет его в переменной mas.

%Открытие текстового файла для чтения (аргумент 'rt') %Имя файла хранится в переменной filename %Идентификатор файла записывается в F

```
F=fopen(filename, 'rt'); if F ~= -1  
mas='';  
while feof(F)== 0  
%Считывание строки  
line=fgetl(F);  
%Добавление считанной строки в массив строк  
mas = char(mas, line);  
end
```

```
%заккрытие файла  
fclose(F);
```

```
%Вывод массива строк в командное окно
```

disp(mas)
end

Обращение к функции: **mm = myview('имя текстового файла')**

Ниже предлагаются возможные имена для функции и ее параметров и вариант программного кода, в котором требуется заменить предложения, написанные на русском языке, предложениями, написанными на МатЛаб-е :

```
function rows = mysearch (filename, substring)
%Напишите комментарии, поясняющие назначение функции
%Открытие файла
F = fopen(filename, 'rt+');
if F ~= 1          %Проверка, успешно ли открыт файл
```

Обнулите счетчик строк, считываемых из файла; Обнулите счетчик строк, содержащих подстроку; Создайте пустой массив для хранения номеров строк;

```
%Последовательная обработка строк в цикле while
while feof(F) == 0
Считывайте          текущую строку из файла;
```

Увеличивайте счетчик считанных строк;

%Сравните длины строки файла и подстроки, так как поиск %имеет смысл в строках, которые длиннее подстроки

```
if ...
```

Определите позицию вхождения подстроки с помощью функции findstr;
if %проверьте, что вхождение имеется

Увеличивайте счетчик найденных строк; Занесите номер найденной строки в выходной массив;

```
end
end
end
fclose(F); % закрытие файла
end
```

Создание словаря из слов текста

```
S=textread('letter.txt', '%s'); %Чтение текста из файла
S,                               %Вывод слов в ячейки в столбик
IS=length(S),                    %Число слов в прочитанном тексте – массиве
ячеек
```

```
%Выделяем из массива любое слово с помощью фигурных скобок и находим его длину
S{5}, l5=length(S{5}),
S{2}, l2=length(S{2}),
```

```
% Убираем знаки препинания, указав параметр delimiter (разделитель слов) и строку signs
с разделителями (пробел, точка, запятая)
signs=' .,';
```

```
S0=textread('letter.txt', '%s', 'delimiter', signs),
```

```
i=7,
```

```
a0=S0{i}, %Выделяем содержимое 7-й ячейки b0=S0{2}, %Вы-
деляем содержимое 2-й ячейки if strcmp(a0, b0), otv=1, else otv=0,
end,
```

```
%Находим самое длинное слово в массиве ячеек-слов
maxword=S{1}; posmaxword=1;
for i=2:1:lS
    if length(maxword) < length(S0{i}) maxword=S0{i}; posmax-
word=i;
end
end
maxword, posmaxword,
disp(['Самое длинное слово      ', maxword]),
```

```
%Есть ли еще слова такой же длины?
j0=posmaxword+1,
for j = j0:1:lS,
    if length (S0{j})== length(maxword)
        disp(['слово ', S0 {j}, ' такой же длины'])
    end
end
```

12. Запись (вывод) в текстовый файл

Информация, которая сохраняется в текстовом файле, представляет собой строки со словами, предложениями или числами. Запись в файл текстовых строк достаточно проста, а что касается чисел, они записываются в текстовый файл с использованием специальных форматов.

Запись строк

Для записи информации в текстовый файл используется оператор **fprintf**. Далее приведены примеры для записи в текстовый файл **строка**. В этих примерах **F** – файловая переменная (идентификатор файла), которая была присвоена ему при открытии командой **fopen**.

```
fprintf(F, 'Строка, записанная в файл командой fprintf. ')
```

Последующая команда `fprintf(F, 'Еще строка. ')` записывает в файл эту строку сразу за предыдущей, а не на новой строке в файле. Для записи текста в файл с новой строки нужно добавить символы перевода строки `\n` в начало новой строки после апострофа:

```
fprintf(F, '\nЭтот текст с новой строки. ')
```

В результате выполнения трех написанных выше команд (одна за другой) содержимое текстового файла будет таким:

Строка, записанная в файл командой `fprintf`. Еще строка. Этот текст с новой строки.

Символы перевода строки `\n` можно также разместить в конце той строки, после которой текст должен начинаться с новой строки.

Вторым аргументом команды **`fprintf`** может быть не только строка, заключенная в апострофы, но и строковая переменная, имеющая некоторое значение.

Форматный вывод числовых данных

Команда **`fprintf`** для вывода числовых данных имеет следующий вид:

```
fprintf(F, 'Список форматов', список переменных)
```

Здесь первый аргумент `F` – идентификатор файловой переменной, как и в случае вывода строк, второй аргумент – строка с кодами форматов, которые определяют вид записи значений переменных из списка, заданного третьим аргументом. Рассмотрим один из применяемых форматов для записи значений вещественных чисел, представленных в виде с фиксированной десятичной точкой. Например, для записи в текстовый файл числа $x=3.1415926$ потребуется всего 9 позиций (с учетом позиции для десятичной точки), из которых 7 позиций понадобятся для записи цифр, стоящих после точки.

Напишите следующую `m`-файл программу и проверьте ее работу.

```
%Программа иллюстрирует форматный вывод в файл чисел, %представ-  
ленных в формате с фиксированной точкой. %Требуется записать значения двух  
переменных x=-pi/4 %и y=sin(x) в файл в формате с фиксированной точкой,  
%оставляя 4 цифры после десятичной точки для x и 8 цифр-для y.
```

```
[F, mes]= fopen('twonum.txt', 'w'); x = -pi / 4; y = sin(x);
```

```
%Пробел(ы) между форматными выражениями разделяют в %файле выводимые  
значения соответствующим числом пробелов
```

```
fprintf(F, '%7.4f %11.8f', x, y); fclose(F);
```

```
%используем написанную ранее функцию myview для вывода %содержимого создан-  
ного файла 'twonum.txt' в командное окно
```

```
myview('twonum.txt');
```

Соответствие форматных выражений и получаемого результата для случая записи форматных выражений (без пробелов) приведено на следующем рисунке.

```
7 . 4 f % 11 . 8 f
```

```
- 0 . 7 8 5 4 - 0 . 7 0 7 1 0 6 7 8
```

Можно сделать запись в файл более наглядной, добавив в форматное выражение соответствующую текстовую строку. Закомментируйте написанный в программе оператор **`fprintf()`** приведенным ниже и посмотрите, как изменится результат:

```
fprintf(F, 'x=%7.4f          y=%11.8f', x, y);
```

Создадим текстовый файл, в который записать информацию об успеваемости студенческой группы (**group**) .

```
function writegroup(filename, group)
```

```
%Использование: writegroup('filename', group)
```

```
N = length (group);    % Нахождение числа студентов в группе
```

```
– Открытие файла с именем filename для записи
```

```
F = fopen(filename, 'wt');
```

```
– Запись шапки таблицы с выравниванием по левому краю каждой строки fprintf(F, '%-10s
```

```
%-10s %-9s %-20s %-8s %-8s\n', 'Фамилия', 'Имя', 'Год',...  

    'Оценки', 'Балл', 'Средний'); %Запись в файл сдержимого полей каж-
```

```
дой структуры в строку
```

```
for i = 1:N
```

```
fprintf(F, '%-10s %-10s %-8.0f %-2.0f %-2.0f %-2.0f %-2.0f %-2.0f...
```

```
%6.0f %-12.4f\n', group(i).Family, group(i).Name,...
```

```
group(i).year, group(i).Marks, group(i).ball, group(i).meanball);
```

```
end
```

```
%Закрытие файла
```

```
fclose(F);function write-
```

```
group(filename, group)  
%файл-функция для записи таблицы с успеваемостью группы %студен-
```

```
тов в текстовый файл.  
%Использование: writegroup('filename', group)
```

```
1 filename - имя файла (в апострофах)
```

```
2 group - массив структур с полями
```

```
3 Family(строка), Name (строка), Year (число),
```

```
4 Marks (вектор-строка с шестью отметками)
```

```
% ball (число) добавленное поле
```

```
% Нахождение числа студентов в группе
```

```
N = length (group);
```

```
% Открытие файла с именем filename для записи
```

```
F = fopen(filename, 'wt');
```

```
% Запись шапки таблицы с выравниванием по левому краю каждой строки fprintf(F, '%-10s %-
```

```
10s %-9s %-20s %-8s %-8s\n', 'Фамилия', 'Имя', 'Год',...  

    'Оценки', 'Балл', 'Средний'); %Запись в файл сдержимого полей каж-
```

```
дой структуры в строку
```

```
for i = 1:N
```

```
fprintf(F, '%-10s %-10s %-8.0f %-2.0f %-2.0f %-2.0f %-2.0f %-2.0f...
```

```
%6.0f %-12.4f\n', group(i).Family, group(i).Name,...
```

```
group(i).year, group(i).Marks, group(i).ball, group(i).meanball);
```

```
end
```

```
%Закрытие файла
```

```
fclose(F);
```