Informatyka

Wykład 3

Witold Dyrka
witold.dyrka@pwr.wroc.pl

19/3/2012

Program wykładów

0. Informatyka. Wprowadzenie do Matlaba (13.02.12)
1. Matlab dla programistów C/C++ (20.02.12)
2. Optymalizacja obliczeń. Grafika (05.03.12)
3. **Złożone typy danych. Programowanie zorientowane obiektowo (OOP)** (19.03.12)
4. OOP część 2 (02.04.12)
5. Graficzny interfejs użytkownika (16.04.12)
6. Obliczenia numeryczne (30.04.12)
7. Kolokwium (14.05.12)

Dzisiejszy wykład w oparciu o...

- B. Mrozek, Z. Mrozek. MATLAB i Simulink. Poradnik użytkownika. Wydanie III. Helion 2010. Rozdział 5.
- MATLAB Product Documentation. Programming and Data Types. http://www.mathworks.com/help/techdoc/ref/f16-42340.html#f16-6840
- MATLAB Product Documentation. Object-Oriented Programming. http://www.mathworks.com/help/techdoc/matlab_oop/ug_intropage.html
  - Example — Representing Structured Data http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html
- Dave Foti. Inside MATLAB Objects in R2008a. Matlab Digest. http://www.mathworks.com/tagteam/50350_91586v00_Digest_OOP_FINAL.pdf
- Scott Gorlin. Advanced Matlab. OOPs, I wrote it again..., Advanced OOP. http://www.scottgorlin.com/category/teaching/

Program na dziś (1)

- **Złożone typy danych**
  - tablica komórkowa (`cell`)
    - zmienna liczba argumentów funkcji
    - przetwarzanie plików i tekstu
  - struktura (`struct`)

- Programowanie zorientowane obiektowo

Ograniczenia zwykłych tablic

- Każda zwykła tablica zawiera jednorodne elementy
  - nie można w jednej tablicy przechowywać
    - **łańcucha** reprezentującego nazwisko
    - oraz **liczby** reprezentującej wiek jednocześnie
- Każda tablica jest prostokątna
  - każdy **wiersz** i **kolumna** muszą mieć równą długość
    - np. w tablicy łańcuchów dopisywaliśmy spacje na końcu
      - mało wygodne i mało uniwersalne
- Rozwiązanie: **kontenery niejednorodne**

Typ komórkowy

- Tablica n-wymiarowa
  - Kontener niejednorodny
    - **Typ komórkowy (`cell`)**
      - tablica elementów różnych typów (n-wymiarowa prostokątna)

`'Jan'`	`'Kowalski'`	`25`
`'Adam'`	`'Nowak'`	`28`

Typ komórkowy (2)

Tworzenie zmiennych typu komórkowego:

- operator **{ }**

```
>> K = {1, rand(5,10,2), speye(3);
        uint8((1:10)'), 'Super!', false}

K =     [          1]    [5x10x2 double]    [3x3 double]
        [10x1 uint8]    'Super!'           [          0]

>> P = {};            % pusta tablica komórkowa 0x0
```

- konstruktor `cell` (puste elementy)

```
>> R = cell(2,3,4); % t. komórkowa 2x3x4 pustych elementów
```

http://www.mathworks.com/help/techdoc/matlab_prog/br04bw6-101.html

Typ komórkowy (3)

```
% K = [          1]  [5x10x2 double]  [3x3 double]
%     [10x1 uint8]  'Super!'          [          0]

>> K{2,4} = R        % R jest tablicą komórkowąą 2x3x4

K =   [          1]  [5x10x2 double]  [3x3 double]  []
      [10x1 uint8]  'Super!'          [          0]  {2x3x4 cell}
```

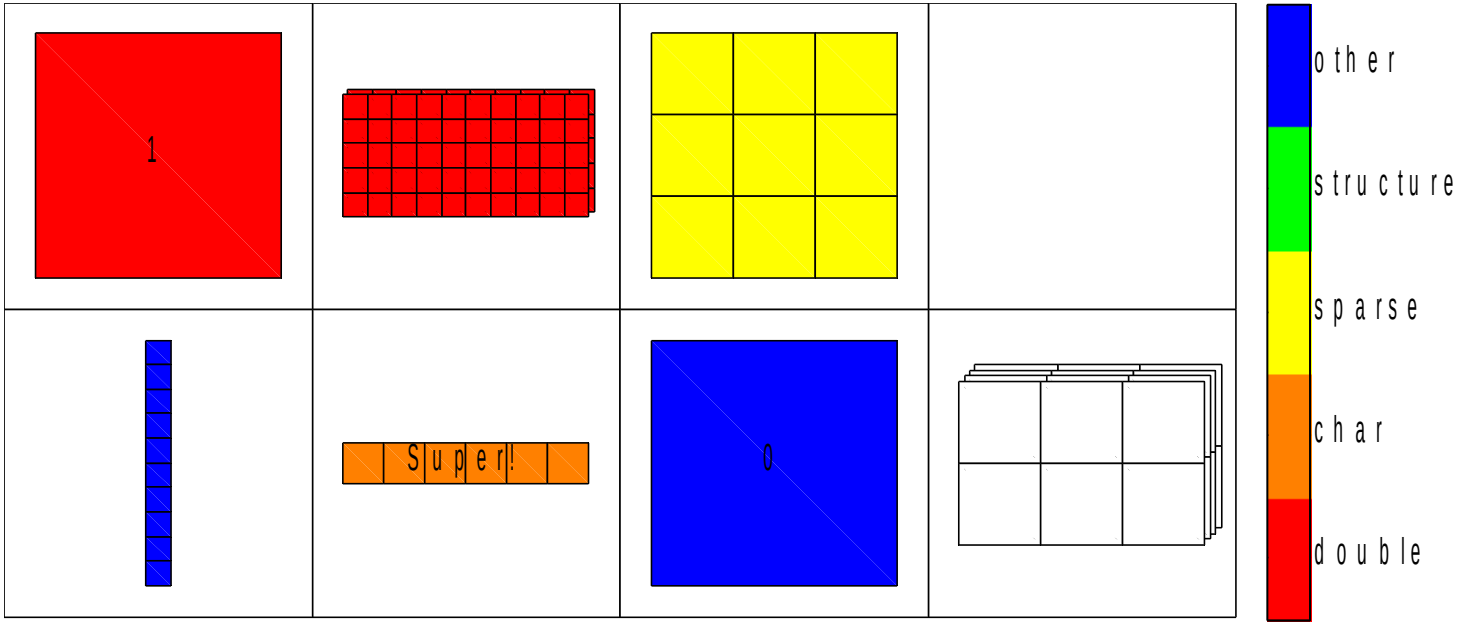
Typ komórkowy (3)

```
% K = [          1]  [5x10x2 double]  [3x3 double]
%     [10x1 uint8]  'Super!'          [          0]

>> K{2,4} = R        % R jest tablicą komórkowąą 2x3x4

K =   [          1]  [5x10x2 double]  [3x3 double]  []
      [10x1 uint8]  'Super!'          [          0]  {2x3x4 cell}

>> cellplot(K)
```



Indeksowanie tablicy komórkowej (1)

```
% K = [          1]  [5x10x2 double]  [3x3 double]  []
%     [10x1 uint8]  'Super!'          [          0]  {2x3x4 cell}
```

- Dostęp do elementów tablicy komórkowej
  - nawiasy klamrowe (ang. *curly braces*) **{ }**

```
>> K{1,3}
ans = (1,1)        1              % wynik jest typu 3x3 double sparse
      (2,2)        1
      (3,3)        1

>> [lancuch, logik] = K{[4,6]}    % zwraca łańcuch i zmienną logiczną
lancuch = Super!
logik = 0

>> K{1,3} = []                    % podst. na poz. 1,3 pustej tablicy
K = [          1]    [5x10x2 double]     []              []
    [10x1 uint8]    'Super!'            [0]    {2x3x4 cell}
```

http://www.mathworks.com/help/techdoc/matlab_prog/br04bw6-117.html

Indeksowanie tablicy komórkowej (2)

```
% K = [          1]  [5x10x2 double]  [3x3 double]  []
%     [10x1 uint8]  'Super!'          [          0]  {2x3x4 cell}
```

- Dostęp do zbioru komórek (podtablicy komórkowej)
  - nawiasy okrągłe (ang. *smooth parentheses*) **( )**

```
>> Klewa = K(:,1:2)            % zwraca tablicę komórkową 2x2
Klewa = [          1]    [5x10x2 double]
        [10x1 uint8]     'Super!'

>> Klewa(1,:) = { 'zmiana1', 'zmiana2' } % modyfikuje 2 komórki
Klewa = 'zmiana1'        'zmiana2'
        [10x1 uint8]     'Super!'

>> Klewa(:) = { 'koniec' }     % modyfikuje wszystkie komórki
Klewa = 'koniec'     'koniec'
        'koniec'     'koniec'

>> K(:,2:3) = []   % usuwa kolumny 2-3 z K
K = [          1]               []
    [10x1 uint8]     {2x3x4 cell}
```

http://www.mathworks.com/help/techdoc/matlab_prog/br04bw6-117.html

Wybrane funkcje operujące na tablicach komórkowych

- **`cell`** ..... tworzy pustą tablicę komórkową
- **`celldisp`** ..... wyświetla zawartość elementów tablicy komórkowej
- **`cellplot`** ..... rysuje strukturę tablicy komórkowej (2-wym.)
- **`mat2cell`** ..... konwertuje zwykłą tablicę na komórkową
- **`num2cell`** ..... konwertuje tablicę numeryczną na komórkową
- **`cell2mat`** ..... konwertuje tablicę komórkową na zwykłą
- **`iscell`** ..... sprawdza, czy zmienna jest tablicą komórkową

Zastosowania tablic komórkowych

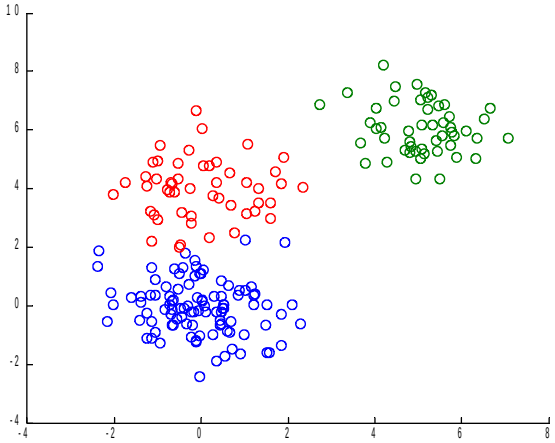
- Obsługa niejednorodnych i nieustrukturyzowanych zbiorów danych, często tymczasowych, np.
  - przekazywanie zmiennej liczby argumentów do i z funkcji
  - przetwarzanie tekstu (np. wczytywanie plików)

Zmienna liczba argumentów funkcji

- Wiele funkcji Matlaba przyjmuje i zwraca zmienną liczbę argumentów, np. `plot`

```
plot(X,Y, 'b-')
plot(X,Y, 'b-', X2,Y2, 'g:')
```

- Podobnie mogą zachowywać się nasze funkcje, np.
  - funkcja licząca środek geometryczny układu punktów




```
[srodek] = srodek_geom(3, X,Y,Z)

[sr1, sr2, sr3] = srodek_geom(2, X,Y, X2,Y2, X3,Y3)
```

Zmienna liczba argumentów funkcji (2)

```
[srodek] = srodek_geom(3, X,Y,Z)
```

zmienna liczba środków

wymiar

zmienna liczba wektorów współrzędnych

```
[sr1, sr2, sr3] = srodek_geom(2, X,Y, X2,Y2, X3,Y3)
```

- Ponieważ rozmiary układów punktów mogą się różnić
  - nie możemy upakować wszystkich współrzędnych w jedną zwykłą tablicę
  - zastosujemy specjalną tablicę komórkową o nazwie `varargin`
    - podobnie listę środków układów zapiszemy w tablicy komórkowej `varargout`

varargin i varargout

varargout = { srodek }

varargin = { X, Y, Z }

```
[srodek] = srodek_geom(3, X,Y,Z)
```

varargout

varargin

```
[sr1, sr2, sr3] = srodek_geom(2, X,Y, X2,Y2, X3,Y3)
```

varargout={sr1,sr2,sr3}

varargin={X,Y,X2,Y2,X3,Y3}

- Nagłówek funkcji:

```
function [varargout] = srodek_geom(wymiar, varargin)
```

Liczba argumentów WE i WY
nargin i nargout

nargout = 1 nargin = 4

```
[srodek] = srodek_geom(3, X,Y,Z)
```

varargout varargin

```
[sr1, sr2, sr3] = srodek_geom(2, X,Y, X2,Y2, X3,Y3)
```

nargout = 3 nargin = 7

Zmienna liczba argumentów funkcji – przykład

```
function [varargout] = srodek_geom(wymiar, varargin)
%SRODEK_GEOM - liczy srodek geometryczny ukladu punktow
% Wywolanie: [srodek] = srodek_geom(1,X)
%            [srodek] = srodek_geom(3,X,Y,Z)
%            [srodek, srodek2, ...] = srodek_geom(2, X,Y, X2,Y2, …)
%  WE: wymiar  - liczba wymiarow przestrzeni
%      X, Y, Z - wektory kolumnowe przechowujace wsp. punktow ukladu
%  WY: srodek  - wektor wierszowy wspolrzednych srodka geometrycznego

% MIEJSCE NA WARUNKI POCZĄTKOWE...

n=0;
for k=1:wymiar:nargin-1    % iteracja po układach punktów, krok co wymiar

    n=n+1;                 % numer kolejny środka

    uklad = cell2mat(varargin(k:k+wymiar-1));    % - pobieramy fragment
                                                 %   tablicy wejściowej
                                                 %   zawierający n-ty ukł.
                                                 % - konwertujemy go
                                                 %   na zwykłą tablicę

    varargout{n} = mean(uklad);   % środek układu współrzędnych
                                  % umieszczamy jako n-ty el. varargout
end
```

Testujemy

```
>> X=rand(100,1); Y=rand(100,1); Z=rand(100,1);
>> X2=rand(50,1)+1; Y2=rand(50,1)+1; Z2=rand(50,1)+1;

>> [s,s2] = srodek_geom(1,X,X2)        % 2x 1 wymiar
s  =  0.4962                           % ok
s2 =  1.4970                           % ok

>> s = srodek_geom(3,X,Y,Z)                   % 3 wymiary
s  =  0.4962    0.5093    0.5315               % ok

>> [s,s2] = srodek_geom(2,X,Y,X2,Y2)  % 2x 2 wymiary
s  =  0.4962    0.5093                 % ok
s2 =  1.4970    1.4844                 % ok

>> s = srodek_geom(3,X',Y',Z')        % podajemy wektory wierszowe
s  =  0.5124                           % co??

>> X3=rand(10)+2; Y3=rand(10)+2;
>> s = srodek_geom(2,X3,Y3)               % podajemy macierze
s  =  Columns 1 through 8
    2.5138    2.6035    2.4296    2.5044    2.7047    2.4634    2.3368    2.5306
  Columns 9 through 16
    2.4061    2.4248    2.5735    2.5161    2.4206    2.4496    2.5301    2.5053
  Columns 17 through 20
    2.5321    2.6542    2.5722    2.3839       % kolejne „nieporozumienie”
```

Poprawki

```
function [varargout] = srodek_geom(wymiar, varargin)
%SRODEK_GEOM - liczy srodek geometryczny ukladu punktow
% ...

if (nargin<1)
    error('Niepodano liczby wymiarow przestrzeni');
end

if mod(nargin-1,wymiar)
    error('Liczba wektor wspolrzednych niezgodna z wymiarem przestrzeni');
end

numer=0;
for k=1:wymiar:nargin-1
    numer=numer+1;
    try                                                 % używamy try-catch
        uklad = cell2mat(varargin(k:k+wymiar-1));       % by dać sensowny
    catch                                               % komunikat błędu
        error('Niezgodny rozmiar wektorow wspolrzednych');
    end

    if size(uklad,2)~=wymiar                        % tutaj załatwiamy nasz
        error('Wymagane wektory kolumnowe');        % problem z wierszowymi
    end                                             % wektorami współrzędnych

    varargout{numer} = mean(uklad);
end
```

Przetwarzanie plików tekstowych

- Bardzo wiele danych jest przechowywanych w plikach tekstowych
  - często łączą dane numeryczne i tekstowe (różnej długości):

```
Janowski   Adam   20     A.Nowak        2012-02-13
Miller     Anna   32     D.Zuch         2011-12-30
```

  - czasami format zmienia się pomiędzy wierszami

```
REMARK  1 PQR file generated by PDB2PQR (Version 1.8)
ATOM  1  N  GLY  1  -8.863  16.944  14.289  0.2943  1.8240
```

- Matlab ma kilka użytecznych funkcji
  - korzystających z tablic komórkowych

Wczytywanie plików tekstowych o jednorodnej strukturze

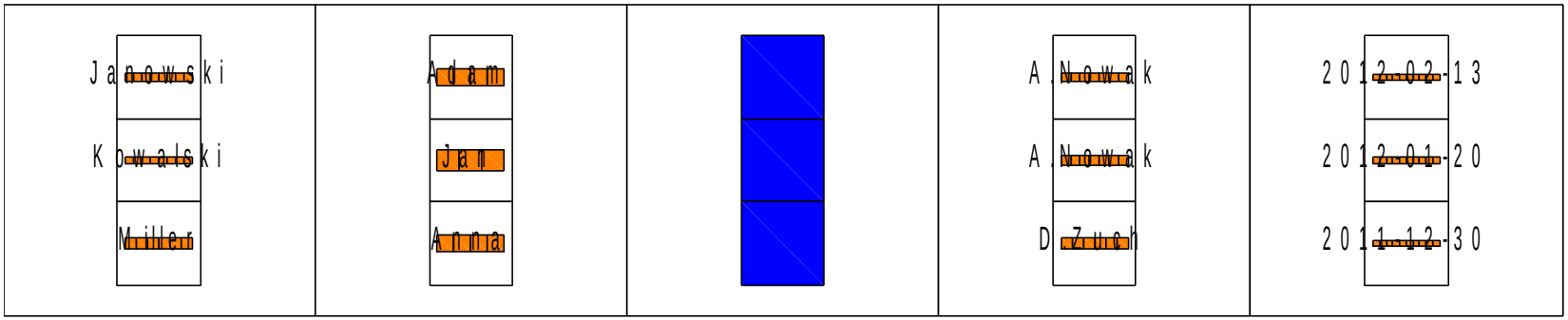
- Tekstowa baza pacjentów
  - dane tekstowe i numeryczne, 1 wiersz nagłówka

Nazwisko	**Imie**	**Wiek**	**Lekarz**	**Ostatnia_wizyta**
Janowski	Adam	20	A.Nowak	2012-02-13
Kowalski	Jan	25	A.Nowak	2012-01-20
Miller	Anna	32	D.Zuch	2011-12-30

```
>> fid = fopen('pacjenci.txt');

>> baza = textscan(fid,'%s %s %d %s %s','HeaderLines',1)
baza =
    {3x1 cell}    {3x1 cell}    [3x1 int32]    {3x1 cell}    {3x1 cell}

>> fclose(fid);
```



http://www.mathworks.com/help/techdoc/ref/textscan.html

Wczytywanie plików tekstowych o niejednorodnej strukturze

- Chcemy wczytać plik PQR reprezentujący strukturę przestrzenną białka
  - dane liczbowe i tekstowe:

```
REMARK   1 PQR file generated by PDB2PQR (Version 1.8)
REMARK   1
REMARK   6 Total charge on this protein: -4.0000 e
REMARK   6
ATOM      1  N    GLY     1      -8.863   16.944   14.289  0.2943 1.8240
ATOM      2  CA   GLY     1      -9.929   17.026   13.244 -0.0100 1.9080
ATOM      3  C    GLY     1     -10.051   15.625   12.618  0.6163 1.9080
ATOM      4  O    GLY     1      -9.782   14.728   13.407 -0.5722 1.6612
ATOM      5  H2   GLY     1      -8.303   17.767   14.241  0.1642 0.6000
ATOM      6  H    GLY     1      -9.296   16.874   15.184  0.1642 0.6000
ATOM      7  H3   GLY     1      -8.304   16.137   14.114  0.1642 0.6000
ATOM      8  HA2  GLY     1     -10.781   17.275   13.670  0.0895 1.1000
ATOM      9  HA3  GLY     1      -9.658   17.675   12.556  0.0895 1.1000
ATOM     10  N    ILE     2     -10.333   15.531   11.332 -0.4157 1.8240
ATOM     11  CA   ILE     2     -10.488   14.266   10.600 -0.0597 1.9080
ATOM     12  C    ILE     2      -9.367   13.302   10.658  0.5973 1.9080
ATOM     13  O    ILE     2      -9.580   12.092   10.969 -0.5679 1.6612
ATOM     14  CB   ILE     2     -10.883   14.493    9.095  0.1303 1.9080
ATOM     15  CG1  ILE     2     -11.579   13.146    8.697 -0.0430 1.9080
ATOM     16  CG2  ILE     2      -9.741   14.794    8.140 -0.3204 1.9080
ATOM     17  CD1  ILE     2     -12.813   13.031    9.640 -0.0660 1.9080
...
```

czytajPQR.m

```
function [nr_at, at, aa, nr_aa, x, y, z, q, r] = czytajPQR(nazwa_pliku)
%CZYTAJPQR - wczytuje strukture bialka z pliku PQR (tylko pola ATOM)
% Wywolanie: [nr_at, at, aa, nr_aa, x, y, z, q, r] =czytajPQR(nazwa_pliku)
% Wejscie: plik w formacie PQR
% Wyjscie:
%   nr_at, at - numer, rodzaj atomu
%   aa, nr_aa - rodzaj, numer aminokwasu, do którego nalezy atom
%   nr_aa - numer aminokwasu, do którego nalezy atom
%   x,y,z - pozycja atomu w przestrzeni
%   q, r  - ladunek, promien atomu

k=0;
fid = fopen(nazwa_pliku,'r');    % otwiera plik do odczytu w trybie bin.

while ~feof(fid)                 % dopóki nie natrafi na koniec pliku EOF
   linia = fgetl(fid);           %   wczytuje linie

   if strncmp(linia,'ATOM',4)    %   jesli linia zaczyna sie
      k = k+1;                   %     od slowa kluczowego ATOM
                                 %     rozbija ja wg formatu PQR

      [nr_at(k), at(k), aa(k), nr_aa(k), x(k), y(k), z(k), q(k), r(k)]=...
          strread(linia, '%*s %d %s %s %d %f %f %f %f %f\n');
   end
end

fclose(fid);                     % zamyka plik PQR
```

Łańcuchy tekstowe zapisano jako tablice komórkowe

```
% *     nr_at  at   aa  nr_aa        x        y        z      q      r
% ATOM      8  HA2  GLY     1     -10.781   17.275   13.670  0.0895 1.1000
% ATOM      9  HA3  GLY     1      -9.658   17.675   12.556  0.0895 1.1000
% ATOM     10  N    ILE     2     -10.333   15.531   11.332 -0.4157 1.8240
% ATOM     11  CA   ILE     2     -10.488   14.266   10.600 -0.0597 1.9080
% ATOM     12  C    ILE     2      -9.367   13.302   10.658  0.5973 1.9080
```

```
>> [nr_at, at, aa, nr_aa, x, y, z, q, r] = czytajPQR('4ins.pqr');
```

- Nazwy atomów (`at`) mają różne długości
  - nie można zapisać ich w tablicy łańcuchów
  - zostają zapisane w tablicy komórkowej

```
>> class(at)
ans = cell
```

- Funkcja `strread` zwraca łańcuchy znaków jako tablice komórkowe
  - podobnie inne funkcje przetwarzające tekst

http://www.mathworks.com/help/techdoc/ref/strread.html

Wczytywanie plików tekstowych i parsowanie tekstu

- dane **numeryczne** + **jednorodna** struktura
  - `load, csvread, dlmread` – zwracają macierz
- dane **alfanumeryczne** + **jednorodna** struktura
  - `textscan` – zwraca tablicę komórkową
  - `textread` – zwraca listę zmiennych
  - `fscanf` – zwraca macierz
- dane **alfanumeryczne** + **niejednorodna** struktura
  - `fgetl` – czyta linia po linii, zwraca łańcuch

  + `strread` – parsowanie łańcucha, zwraca listę zmiennych
  + `textscan` – parsowanie łańcucha, zwraca tablicę komórkową

Struktura

- Tablica n-wymiarowa
  - Kontener niejednorodny
    - struktura (`struct`)
      - składa się z nazwanych pól różnych typów
      - częściowo podobna do typu **struct** w C

- Zastosowanie
  - organizowanie danych, strukturyzacja
    - bardzo użyteczne, patrz projekt z *Języków programowania*
  - reprezentacja znaczenia danych
    - często czytelniejszy kod

http://www.mathworks.com/help/techdoc/ref/struct.html

Tworzenie struktur (1)

```
>> Pacjent.Nazwisko = 'Janowski';    % przez przypisanie
>> Pacjent.Imie = 'Adam';            % wartości do pola
>> Pacjent.Wiek = 20;

>> Pacjent =                         % przez konstruktor
   struct('Nazwisko','Janowski','Imie','Adam','Wiek',20);

>> Pacjent(1)    % ≡ >>Pacjent, gdy tylko jedna struktura
ans = Nazwisko: 'Janowski'
      Imie: 'Adam'
      Wiek: 20

>> Pacjent(2) =        % dodawanie kolejnego pacjenta
   struct('Nazwisko','Kowalski','Imie','Jan','Wiek',25);

>> Pacjent
Pacjent = 2x1 struct array with fields:
    Nazwisko
    Imię
    Wiek

>> Pacjent(2)
ans =  Nazwisko: 'Kowalski'
       Imie: 'Jan'
       Wiek: 25
```

http://www.mathworks.com/help/techdoc/ref/struct.html

Tworzenie struktur (2)

```
% Nazwisko  Imie    Wiek    Lekarz          Ostatnia_wizyta
% Janowski  Adam    20      A.Nowak         2012-02-13
% Kowalski  Jan     25      A.Nowak         2012-01-20
% Miller    Anna    32      D.Zuch          2011-12-30

>> fid = fopen('pacjenci.txt');
>> baza = textscan(fid,'%s %s %d %s %s','HeaderLines',1)
baza =
    {3x1 cell}    {3x1 cell}    [3x1 int32]    {3x1 cell}    {3x1 cell}
>> fclose(fid);

>> Pacjent =       % tworzenie całej tablicy struktur z tablic komórkowych
struct('Nazwisko',baza{1},'Imie',baza{2},'Wiek',num2cell(baza{3}));
Pacjent = 3x1 struct array with fields:
    Nazwisko
    imię
    Wiek
```

http://www.mathworks.com/help/techdoc/ref/struct.html

Odwołania do pól struktury

- przez nazwę pola

```
>> Pacjent(1).Nazwisko
ans = Janowski
>> [p1, p2, p3] = Pacjent.Nazwisko
p1 = Janowski
p2 = Kowalski
p3 = Miller
```

- dynamiczne
  - nazwa pola jest zmienną

```
>> pole = 'Wiek'
>> Pacjent(2).(pole)
ans =  20

>> pole = 'wiek';              % Uwaga na wielkość liter !
>> Pacjent(2).(pole)
??? Reference to non-existent field 'wiek'.
```

http://www.mathworks.com/help/techdoc/learn_matlab/f4-2137.html#f4-2429

Wybrane funkcje operujące na strukturach

- **`struct`** ..... tworzy strukturę
- **`fieldnames`** ..... zwraca nazwy pól
- **`getfield,setfield`** ... pobiera/ustawia wartość pola
- **`rmfield`** ..... usuwa pole
- **`num2cell`** ..... konwertuje tablicę numeryczną na komórkową
- **`isfield`** ..... sprawdza czy pole istnieje
- **`isstruct`** ..... sprawdza, czy zmienna jest strukturą
- **`struct2cell`** ..... zamienia tablicę struktur na tablicę komórkową
- **`cell2struct`** ..... zamienia tablicę komórkową na tablicę struktur

Ograniczenia struktur

- Struktury pozwalają organizować dane
  - oraz reprezentować ich znaczenie

- Niestety bardzo słabo chronią przed błędami
  - można niechcący dopisać nowe pole

```
>> Pacjent(1).Weik = 21;
>> Pacjent(1)
ans =  Nazwisko: 'Janowski'
       Imie: 'Adam'
       Wiek: 20
       Weik: 21
```

  - nie można ograniczyć dostępu do danych

Program na dziś (2)

- Złożone typy danych

- **Programowanie zorientowane obiektowo**
  - tworzenie klas w Matlabie
  - własności i metody w Matlabie
  - enkapsulacja

Uwaga – wymaga MATLABa R2008a (7.6) lub nowszego

Programowanie zorientowane obiektowo

Object-oriented programming (OOP)

- W zasadzie zaczęliśmy już programować „obiektowo” na kursie *Języki programowania*
  - nauczyliśmy się tworzyć proste klasy reprezentujące pojęcia i byty świata „rzeczywistego”
    - tokeny, strumienie, daty (na wykładzie)
    - pacjentów, karty medyczne itp. (w ramach projektu)
- Wiemy, że <u>klasa</u> określa
  - przechowywane dane i dozwolone operacje na nich
  - *obiekt jest małżeństwem danych i operacji* (Scott Gorlin)
- Dziś przeniesiemy tę wiedzę na grunt Matlaba

http://www.mathworks.com/help/techdoc/matlab_oop/brhzttf.html

Typowy przepływ pracy (*workflow*)

- Analiza problemu
  - co jest obiektem?
  - jakie **cechy obiektu** są istotne dla danego problemu?
  - co może się **dziać z obiektem**?

- Projekt
  - **dane** i **stany** obiektu, które należy przechowywać
  - **operacje**, które będą wykonywane na obiekcie

- Implementacja
- Testowanie

http://www.mathworks.com/help/techdoc/matlab_oop/brhzttf.html

Przykład – analiza problemu

- Problem:
  - Liczenie modułu sprężystości (np. materiału na protezę)
- Co jest obiektem?
  - seria pomiarów
    - naprężenie
    - odkształcenie
- Co się może dziać z obiektem?
  - wprowadzanie pomiarów
  - wyświetlanie pomiarów
  - obliczanie modułu sprężystości
  - wykreślanie krzywej

Przykład – projekt klasy

- klasa `PomiarElastycznosci`
  - dane (własności, ang. `properties`) % czyli zmienne
    - `Material` % łańcuch znaków określający testowany materiał
    - `NumerProbki` % numer próbki materiału
    - `Naprezenie` % wektor liczb reprezentujących naprężenie
      % podczas testu
    - `Odksztalcenie` % wektor liczb reprezentujących odkształcenie
      % podczas testu, odpowiednio do wektora `Sila`
    - `Modul` % moduł sprężystości Younga
  - operacje (metody, ang. `methods`) % czyli funkcje
    - `PomiarElastycznosci` % konstruktor
    - `wyswietl` % wyświetlanie danych
    - `liczModul` % obliczanie modułu sprężystości
    - `rysuj` % rysowanie krzywej

Definiowanie własności klasy

```
classdef PomiarElastycznosci          % classdef zamiast class

    %POMIARELASTYCZNOSCI przechowuje pomiary elastycznosci
    % i liczy modul Younga
    %   ...

    properties                        % wszystkie dane w sekcji
                                      % properties

      Material = '';                  % zmienne można inicjować

      NumerProbki = 0;

      Naprezenie

      Odksztalcenie

      Modul = 0;

    end

end
```

http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html

Obiekt jako struktura o określonych polach

```
pe = PomiarElastycznosci;                % pusty obiekt
pe.Material = 'stal niestopowa';
pe.NumerProbki = 001;
pe.Naprezenie = [2e4 4e4 6e4 8e4];
pe.Odksztalcenie = [.12 .20 .31 .40];
pe.Modul = mean(pe.Naprezenie./pe.Odksztalcenie);
```

```
>> pe
pe = PomiarElastycznosci
  Properties:
        Material: 'stal niestopowa'
     NumerProbki: 1
      Naprezenie: [20000 40000 60000 80000]
   Odksztalcenie: [0.1200 0.2000 0.3100 0.4000]
           Modul: 1.9005e+005
  Methods
```

```
>> pe.Materia = 'stal niestopowa'    % literówka? Nie przejdzie !
No public field Materia exists for class PomiarElastycznosci.
```

```
>> class(pe)
ans = PomiarElastycznosci                % Ok, identyfikacja typu
```

Konstruktor

- Wpisywanie pole po polu jest niewygodne i błędogenne
  - napiszmy konstruktor
    - czyli funkcję tworzącą obiekt
      - konstruktor nosi taką samą nazwę jak klasa

warunki początkowe

```
function pe = PomiarElastycznosci(material,numprob,naprez,odkszt)

    if nargin~=4  % Sprawdzamy poprawną liczbę danych
        error('Wymagane 4 argumenty konstruktora...');
    elseif length(odkszt)>length(naprez)  % oraz zgodnosc pomiarow
                                          % odksztalcenia i naprezenia
        error('Niezgodne liczby pomiarów odksztalcenia i naprezenia!');
    end

    pe.Material = material;
    pe.NumerProbki = numprob;
    pe.Naprezenie = naprez;
    pe.Odksztalcenie = odszkt;

end % PomiarElastycznosci
```

Konstruktor (2)

```
>> pe = PomiarElastycznosci('stal niestopowa',001,[2e4 4e4 6e4 8e4],...
                            [.12 .20 .31 .40])

pe = PomiarElastycznosci
  Properties:
        Material: 'stal niestopowa'
     NumerProbki: 1
      Naprezenie: [20000 40000 60000 80000]
   Odksztalcenie: [0.1200 0.2000 0.3100 0.4000]
           Modul: 0
  Methods
```

```
>> pe = PomiarElastycznosci('stal niestopowa',001,[2e4 4e4 6e4 8e4])
Error using PomiarElastycznosci (line 20)
Wymagane 4 argumenty konstruktora
```

```
>> pe = PomiarElastycznosci('stal niestopowa',001,[2e4 4e4],[.12 .20 .31])
Error using PomiarElastycznosci (line 22)
Niezgodne liczby pomiarów odksztalcenia i naprezenia!
```

Enkapsulacja

- Oddzielenie i ukrycie wnętrzności klasy
  - czyli **implementacji** (*jak to działa?*)
  - od powierzchowności klasy
    - czyli **interfejsu** (*jak z tego korzystać?*)

- Ułatwia zachowanie poprawności obiektu
  - ogranicza dostęp do danych
- Umożliwia ulepszanie sposobu działania klasy
  - bez zmiany sposobu korzystania z niej

- Podstawowa zasada programowania zorientowanego obiektowo

Enkapsulacja (2)

- Enkapsulacja (zwana też hermetyzacją)
  - jest jedną podstawowych zasad inżynierii,
    - odporność na błędy
    - modularność
    - kompatybilność

Enkapsulacja w C++

- W obiektowym C++ lub Javie (zasadniczo)
  - dane (i niektóre funkcje) są prywatne
  - dane (i niektóre funkcje) są dostępne tylko przez funkcje publiczne

- Mamy kontrolę nad poprawnością obiektu (niezmienniki)
- Chcemy zmienić reprezentację danych w kolejnej wersji?
  - dostosowujemy sposób działania publicznych funkcji
  - nie modyfikujemy nagłówków funkcji
    - korzystanie z nich nie ulega zmianie

Enkapsulacja w Matlabie

- W Matlabie też możemy ograniczyć dostęp do danych
  - ale czasem to niewygodne i nieczytelne
    - gdy operujemy na tablicach
      - np. taki prosty kod

```
x.Data(1:2:end, 1:2:end) = y
```

      - wyglądałby tak, gdyby `Data` było prywatne:

```
temp = x.getData;
temp(1:2:end, 1:2:end) = y
x.setData(temp)
```

http://www.mathworks.com/tagteam/50350_91586v00_Digest_OOP_FINAL.pdf

Enkapsulacja w Matlabie (2)

- Idziemy na kompromis
  - ukrywanie danych – nie
  - sprawdzanie poprawności – tak
    - jeśli zdefiniujemy dla danej funkcje `set` i `get`
      - Matlab wywoła je, gdy chcemy zmienić/odczytać wartość danej

Zaraz zobaczymy to na naszym przykładzie

Sprawdzanie poprawności danych

- Założenie
  - mierzymy elastyczność elementów z trzech rodzajów metalu:
    - stali niestopowej, stali nierdzewnej i aluminium
  - nie chcemy bałaganu w danych
    - wymagamy by użytkownik stosował nazwy `'stal niestopowa'`, `'stal nierdzewna'` i `'aluminium'`
  - potrzebujemy metody weryfikacji
    - przy ustawianiu własności

```
methods
  function obj = set.Material(obj,material)

     % Pseudokod:
     % jeśli niewłaściwy material, zgłoś błąd
     % jeśli prawidłowy ustaw go, czyli: obj.Material = material

  end % set.Material
end % methods
```

```
>> pe.Material = 'olow'
Error using PomiarElastycznosci/set.Material (line 17)
Akceptowany material to aluminium, stal nierdzewna, stal niestopowa
```

Weryfikacja danych (2)

- Założenie
  - mierzymy elastyczność elementów z trzech rodzajów metalu:
    - stali niestopowej, stali nierdzewnej i aluminium
  - nie chcemy bałaganu w danych
    - wymagamy by użytkownik stosował nazwy `'stal niestopowa'`, `'stal nierdzewna'` i `'aluminium'`
  - potrzebujemy metody weryfikacji:

```
methods
  function obj = set.Material(obj,material)
     if ~(strcmpi(material,'aluminum') ||...
        strcmpi(material,'stal nierdzewna') ||...
        strcmpi(material,'stal niestopowa'))
        error('Akceptowany material to aluminium, stal nierdzewna,
               stal niestopowa')
     end
  obj.Material = material;
  end % set.Material
end % methods
```

```
>> pe.Material = 'olow'
Error using PomiarElastycznosci/set.Material (line 17)
Akceptowany material to aluminium, stal nierdzewna, stal niestopowa
```

http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html

Liczenie modułu sprężystości

- Moduł sprężystości łatwo policzyć na podstawie pomiarów (jest od nich *zależny*):

  `Modul = Naprezenie / Odksztalcenie`

- Wygodnie liczyć go używając funkcji `get`

```
function modul = get.Modul(obj)

    ind = find(obj.Odksztalcenie > 0); % Tylko niezerowe odkształcenie
    modul = mean(obj.Naprezenie(ind)./obj.Odksztalcenie(ind));

end % get.Modul
```

Liczenie modułu sprężystości (2)

- Moduł sprężystości łatwo policzyć na podstawie pomiarów (jest od nich *zależny*):

```
Modul = Naprezenie / Odksztalcenie
```

- Wartości modułu nie trzeba przechowywać w obiekcie
  - może ulec zmianie np. gdy dodamy nowe pomiary

```
properties (Dependent = true) % własności zależne
    Modul
end
```

- Użytkownik nie powinien niechcący nadpisać wartości `Modul`-u

```
properties (Dependent = true, SetAccess = private)
    Modul
end
```

```
>> pe.Modul = 1
Setting the 'Modul' property of the 'PomiarElastycznosci' class is
not allowed.
```

http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html

Tak to wygląda teraz

```
classdef PomiarElastycznosci
    %POMIARELASTYCZNOSCI przechowuje pomiary elastycznosci ...

    properties
      Material
      NumerProbki
      Odksztalcenie
      Naprezenie
    end % properties

    properties (Dependent = true, SetAccess = private)
       Modul
    end % Dependent + set-private properties

    methods
      function pe = PomiarElastycznosci(material,numprob,naprez,odkszt) % ...

      function obj = set.Material(obj,material)
         if ~(strcmpi(material,'aluminum') ||...
            strcmpi(material,'stal nierdzewna') ||...
            strcmpi(material,'stal niestopowa'))
            error('Akceptowany material to aluminium...')
         end
      obj.Material = material;
      end % set.Material

      function modul = get.Modul(obj)
        ind = find(obj.Odksztalcenie > 0); % Tylko niezerowe odksztalcenie
        modul = mean(obj.Naprezenie(ind)./obj.Odksztalcenie(ind));
      end % get.Modul

    end % methods

end
```

Kilka zasad

- **Własności** umieszczamy w blokach `properties`
  - osobne bloki dla własności o różnych atrybutach
- **Metody** umieszczamy w blokach `methods`
  - **konstruktor** tworzy obiekt, nazywa się tak jak klasa

```
function pe = PomiarElastycznosci(material,numprob,naprez,odkszt)
```

  - funkcja `get` jest uruchamiana gdy czytamy daną
    - parametrem funkcji `get` jest obiekt, zwracana jest wartość własności

```
function modul = get.Modul(obj)
```

  - funkcja `set` jest uruchamiana gdy zmieniamy daną
    - parametrami funkcji `set` są obiekt i nowa wartość, zwracany jest obiekt

```
function obj = set.Material(obj, material)
```

http://www.mathworks.com/help/techdoc/matlab_oop/brgsek9-1.html

Nierozwiązany problem

- Konstruktor sprawdzał czy nie podano więcej wartości odkształcenia niż naprężenia
- Metoda `set`, np. dla `Odksztalcenie`, nie powinna sprawdzać długości wektora `Naprezenie`
  - nie ma gwarancji kolejności w jakiej Matlab ładuje obiekt
- Możemy
  - **albo** określić `Odksztalcenie` i `Naprezenie` jako prywatne
    - rezygnujemy z wygodnego przypisywania
  - **albo** zastosować nieco skomplikowane obejście problemu
  - **albo** pogodzić się z potencjalnym błędem:

```
>> pe.Odksztalcenie = [.12 .20 .31 .40 .50];
>> pe.Naprezenie = [2e4 4e4 6e4 8e4];
>> pe.Modul
Index exceeds matrix dimensions.
Error in PomiarElastycznosci/get.Modul (line 44)
  modul = mean(obj.Naprezenie(ind)./obj.Odksztalcenie(ind));
```

Projekt klasy – co nam zostało?

- klasa `PomiarElastycznosci`
  - dane (`properties`)
    - gotowe
  - operacje (`methods`)
    - `PomiarElastycznosci` % konstruktor - gotowe
    - `get.Modul` % obliczanie modułu - gotowe
    - `set.Material` % weryfikacja materiału - dodane i gotowe
    - `wyswietl` % wyświetlanie danych - do zrobienia
    - `rysuj` % rysowanie krzywej - do zrobienia

Wyświetlanie obiektu

- W Matlabie, do tekstowego wyświetlania danych, służy funkcja `disp`
- Napiszemy funkcję `disp` dla danych typu `PomiarElastycznosci`

```
function disp(obj)
%DISP wyswietla obiekt typu PomiarElastycznosci
% Wywolanie: disp(obj)
%   obj - obiekt typu PomiarElastycznosci

    fprintf(1,'Material: %s\nNumerProbki: %g\nModul:%1.5g\n',...
              obj.Material,obj.NumerProbki,obj.Modul);

               % 1 oznacza standardowe wyjście
end % disp
```

```
>> disp(pe)
Material: stal niestopowa
NumerProbki: 1
Modul:1.9005e+005

>> pe
Material: stal niestopowa
NumerProbki: 1
Modul:1.9005e+005
```

http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html

Przeładowanie funkcji (*overloading*)

- Właśnie przeładowaliśmy funkcję `disp` dla naszej klasy
  - tzn. stworzyliśmy jej wersję specjalną
    - uruchamianą dla typu `PomiarElastycznosci`

```
>> help disp
 disp Display array.
    disp(X) displays the array, without printing the array name.  In
    all other ways it's the same as leaving the semicolon off an
    expression except that empty arrays don't display.

    If X is a string, the text is displayed.
    See also int2str, num2str, sprintf, rats, format.

    Overloaded methods:
       opaque/disp
       PomiarElastycznosci/disp
       inline/disp
       ...

>> help PomiarElastycznosci/disp
 disp wyswietla obiekt typu PomiarElastycznosci
  Wywolanie: disp(obj)
    obj - obiekt typu PomiarElastycznosci
```

Rysowanie krzywej naprężenia

- W Matlabie, do rysowania wykresów służy funkcja `plot`
- Napiszemy funkcję `plot` dla danych typu `PomiarElastycznosci`

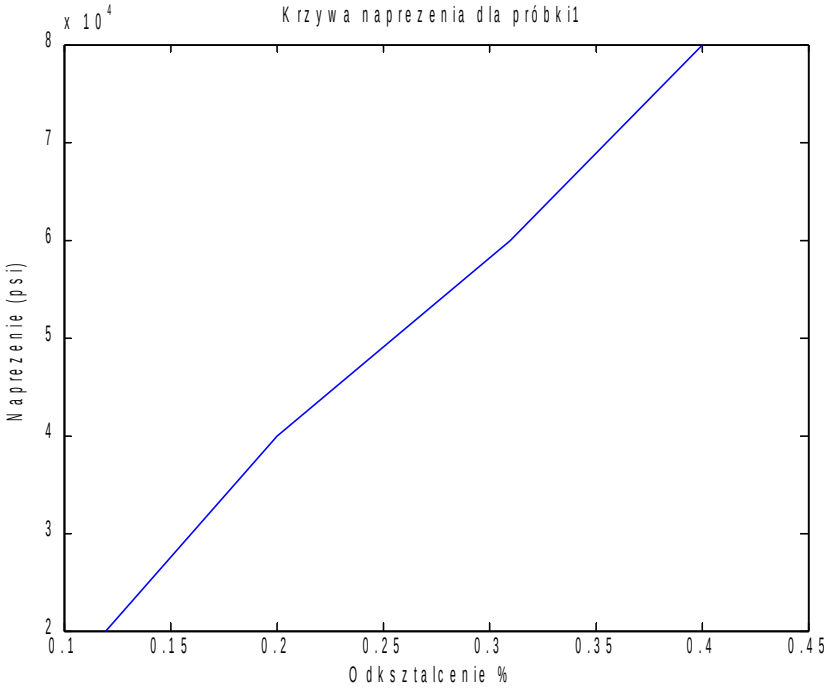
```
function plot(obj,varargin)
%PLOT rysuje wykres krzywej naprezenia

    plot(obj.Odksztalcenie,obj.Naprezenie,varargin{:}) % "zwykly" plot

    title(['Krzywa naprezenia dla próbki',...
        num2str(obj.NumerProbki)])

    ylabel('Naprezenie (psi)')
    xlabel('Odksztalcenie %')

end % plot
```



http://www.mathworks.com/help/techdoc/matlab_oop/f2-74190.html

Wersja 1 – gotowe

- klasa `PomiarElastycznosci`
  - dane (`properties`)
    - gotowe
  - operacje (`methods`)
    - `PomiarElastycznosci` % konstruktor - gotowe
    - `get.Modul` % obliczanie modułu - gotowe
    - `set.Material` % weryfikacja materiału - dodane i gotowe
    - `disp` % wyświetlanie danych - gotowe
    - `plot` % rysowanie krzywej - gotowe

Kilka trików

- Obiekty można zapisywać i ładować:

```
>> save 'moj_obiekt' pe, clear pe, load 'moj_obiekt' pe
pe =  Material: stal niestopowa
      NumerProbki: 1
      Modul:1.9005e+005
```

- Po modyfikacji własności oraz po dodaniu metody
  - należy usunąć wszystkie obiekty danej klasy z pamięci
    - aby zmiany odniosły skutek

```
>> clear pe
```

- Wyświetlanie metod klasy:

```
>> methods PomiarElastycznosci
Methods for class PomiarElastycznosci:
PomiarElastycznosci  disp                    plot
```

Dziś najważniejsze było...

- Tablice komórkowe i ich zastosowanie
  - dostęp do tablic komórkowych
  - varargin i varargout
- Czytanie danych z plików tekstowych
- Programowanie zorientowane obiektowo
  - tworzenie klas w Matlabie
  - enkapsulacja – naturalna potrzeba inżyniera

A za 2 tygodnie...

- Programowanie zorientowane obiektowo – cd
  - dziedziczenie
  - polimorfizm
  - uchwyty