

# Informatyka 1 (ES1F1002)

---

Politechnika Białostocka - Wydział Elektryczny  
Elektrotechnika, semestr II, studia stacjonarne I stopnia  
Rok akademicki 2022/2023

**Wykład nr 9 (05.12.2022)**

dr inż. Jarosław Forenc

## Plan wykładu nr 9

- Tablice w języku C
  - tablice wielowymiarowe
- Tablice o zmiennym rozmiarze (VLA)
- Łańcuchy znaków w języku C
  - implementacja, deklaracja, inicjalizacja
  - stała znakowa
  - wyświetlenie i wczytanie tekstu
  - plik nagłówkowy string.h
- Struktury, pola bitowe, unie
  - deklaracja struktury i zmiennej strukturalnej
  - odwołania do pól struktury
  - inicjalizacja zmiennej strukturalnej
  - złożone deklaracje struktur

# Język C - tablice wielowymiarowe

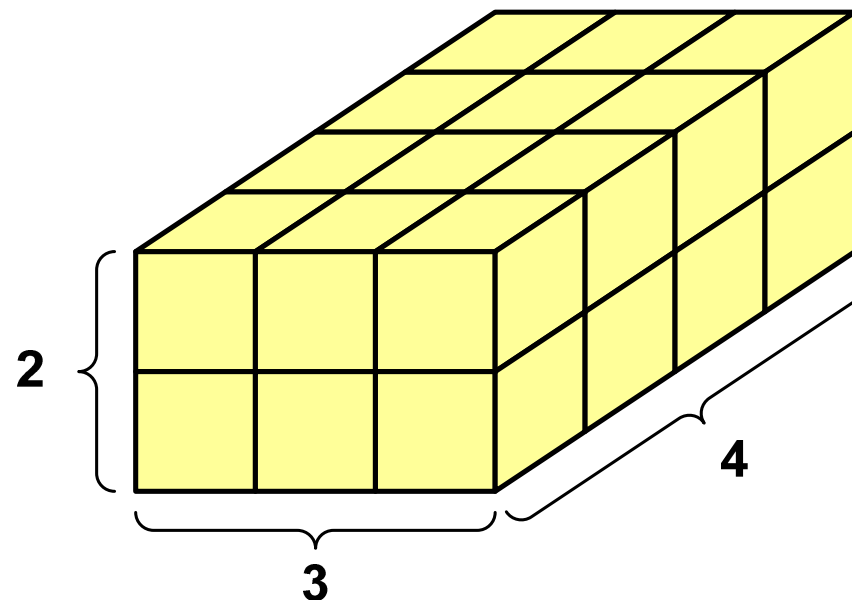
- Deklaracja tablicy wielowymiarowej

**typ nazwa [wymiar\_1] [wymiar\_2]... [wymiar\_N]**

- Deklaracja tablicy trójwymiarowej

```
int tab[4][2][3];
```

- Inicjalizacja i odwoływanie się do elementów są analogiczne jak w przypadku macierzy



# Język C - tablice wielowymiarowe

```
#include <stdio.h>
```

```
#define X 3
```

```
#define Y 2
```

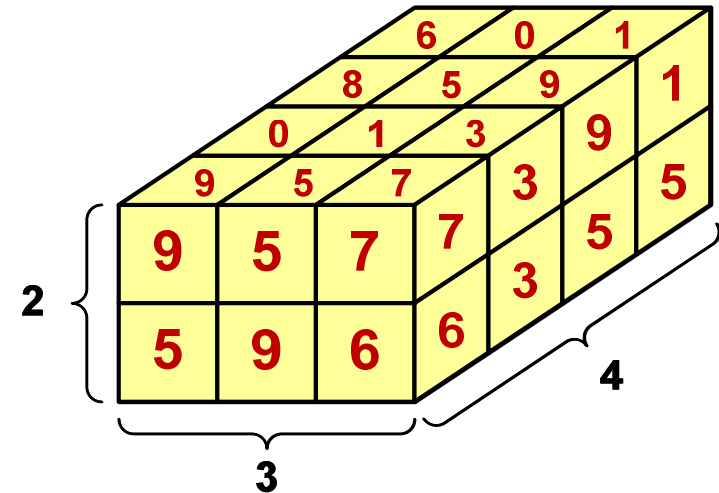
```
#define Z 4
```

```
int main(void)
```

```
{
```

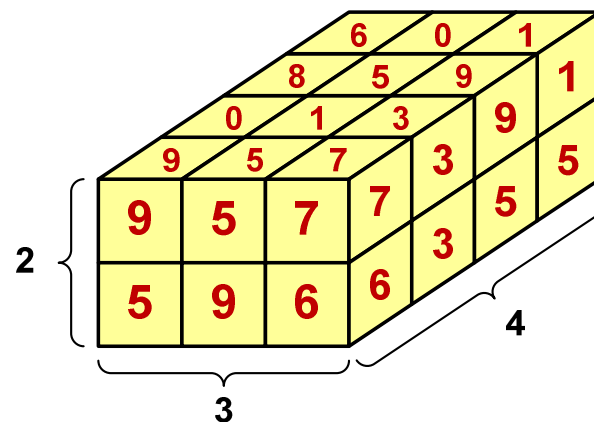
```
    int x, y, z;
```

```
    int tab[Z][Y][X] = {{{9, 5, 7}, {5, 9, 6}},  
                        {{0, 1, 3}, {7, 4, 3}},  
                        {{8, 5, 9}, {1, 3, 5}},  
                        {{6, 0, 1}, {8, 2, 5}}};
```



# Język C - tablice wielowymiarowe

```
for (z=0; z<Z; z++)  
{  
    for (y=0; y<Y; y++)  
    {  
        for (x=0; x<X; x++)  
            printf("%3d", tab[z][y][x]);  
        printf("\n");  
    }  
    printf("\n");  
}  
  
return 0;  
}
```



9	5	7
5	9	6
0	1	3
7	4	3
8	5	9
1	3	5
6	0	1
8	2	5

## Język C - tablice o zmiennym rozmiarze (VLA)

- **VLA** (ang. variable length array) - tablice, których rozmiar określany jest na etapie wykonywania programu (np. jako rozmiar może wystąpić nazwa zmiennej)

```
int n;  
n = 10;  
int T[n];
```

```
int n;  
scanf("%d", &n);  
int T[n];
```

- Rozmiar tablicy, a standardy języka C:
  - do standardu C99 rozmiar tablicy musiał być stałym wyrażeniem całkowitym (stała liczbowa: `5`, `#define N 5`, `const int n = 5;`)
  - w standardzie C99 wprowadzono tablice o zmiennym rozmiarze
  - w standardzie C11 tablice o zmiennym rozmiarze określone są jako opcjonalne dla implementacji

## Język C - tablice VLA (Visual Studio 2008 / 2019)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int n, i;

    printf("Rozmiar wektora: ");
    scanf("%d", &n);

    float T[n];

    for (i=0; i<n; i++)
        T[i] = sqrt((float)i);

    for (i=0; i<n; i++)
        printf("T[%d] = %f\n", i, T[i]);

    return 0;
}
```

## Język C - tablice VLA (Visual Studio 2008 / 2019)

```
#include <stdio.h>
#include <math.h>
```

```
int main(void)
{
```

```
    int n, i;
```

```
    printf("Rozmiar wektora: ");
    scanf("%d", &n);
```

```
    float T[n];
```

```
    for (i=0; i<n; i++)
        T[i] = sqrt((float)i);
```

```
    for (i=0; i<n; i++)
        printf("T[%d] = %f\n", i, T[i]);
```

```
    return 0;
```

```
}
```

error C2057: expected constant expression  
error C2466: cannot allocate an array of constant size 0  
error C2133: 'T' : unknown size

error C2057: oczekiwano stałego wyrażenia  
error C2466: nie można przydzielić tablicy stałego rozmiaru 0  
error C2133: "T": nieznan rozmiar



## Język C - tablice VLA (Dev-C++, Code::Blocks)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int n, i;

    printf("Rozmiar wektora: ");
    scanf("%d", &n);

    float T[n];

    for (i=0; i<n; i++)
        T[i] = sqrt((float)i);

    for (i=0; i<n; i++)
        printf("T[%d] = %f\n", i, T[i]);

    return 0;
}
```

```
Rozmiar wektora: 8
T[0] = 0.000000
T[1] = 1.000000
T[2] = 1.414214
T[3] = 1.732051
T[4] = 2.000000
T[5] = 2.236068
T[6] = 2.449490
T[7] = 2.645751
```

## Język C - tablice VLA

- Tablica VLA może być także tablicą dwu- lub wielowymiarową

```
int n = 5, m = 6;  
int T1[n][m], T2[n][m][n];
```

- Nie można modyfikować rozmiaru tablic VLA po deklaracji
- Tablice VLA nie mogą być inicjalizowane podczas deklaracji
  - błędy i ostrzeżenia w `Code::Blocks`

```
error: variable-sized object may not be initialized  
warning: excess elements in array initializer  
warning: (near initialization for 'T')
```

- w `Dev-C++` inicjalizacja jest dopuszczalna!

## Język C - łańcuchy znaków

- **łańcuch znaków** (ciąg znaków, napis, literał łańcuchowy, stała łańcuchowa, C-string) - ciąg złożony z zera lub większej liczby znaków zawartych między znakami cudzysłowu

"Pies"

- Implementacja - tablica, której elementami są pojedyncze znaki (typ **char**)

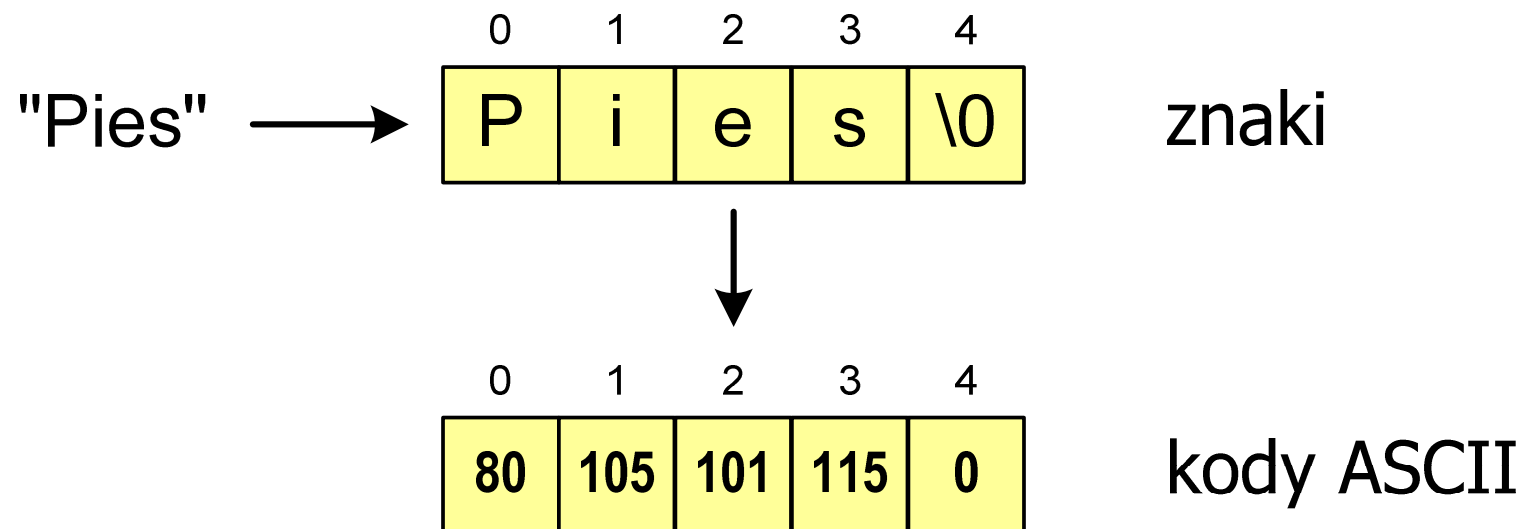
"Pies" → 

0	1	2	3	4
P	i	e	s	\0

- Ostatni znak (**\0**, liczba **zero**, znak zerowy) oznacza koniec napisu

## Język C - łańcuchy znaków

- W rzeczywistości w tablicy zamiast znaków przechowywane są odpowiadające im kody ASCII (czyli liczby)



## Język C - deklaracja łańcucha znaków

- Deklaracja zmiennej przechowującej łańcuch znaków

```
char nazwa_zmiennej[rozmiar];
```

Przykład:

```
char txt[10];
```

- Tablica `txt` może przechowywać napisy o maksymalnej długości do 9 znaków

## Język C - inicjalizacja łańcucha znaków

- Inicjalizacja łańcucha znaków

```
char txt1[10] = "Pies";  
char txt2[10] = {'P', 'i', 'e', 's'};  
char txt3[10] = {80, 105, 101, 115};
```

- Pozostałe elementy tablicy otrzymują wartość zero

P	i	e	s	\0	\0	\0	\0	\0	\0
---	---	---	---	----	----	----	----	----	----

```
char txt4[] = "Pies";  
char *txt5 = "Pies";
```

## Język C - inicjalizacja łańcucha znaków

- Inicjalizacja możliwa jest tylko przy deklaracji

```
char txt[10];  
txt = "Pies";    /* BŁĄD!!! */
```

- Przypisanie zmiennej `txt` wartości `"Pies"` wymaga zastosowania funkcji `strcpy()` z pliku nagłówkowego `string.h`

```
char txt[10];  
strcpy(txt, "Pies");
```

## Język C - stała znakowa

- **Stałą znakową** tworzy jeden znak ujęty w apostrofy

```
char zn = 'x';
```

- W rzeczywistości stała znakowa jest to liczba całkowita, której wartość odpowiada wartości kodu ASCII reprezentowanego znaku
- Zamiast powyższego kodu można napisać:

```
char zn = 120;
```

- Uwaga:
  - **'x'** - stała znakowa (jeden znak)
  - **"x"** - łańcuch znaków (dwa znaki: x oraz \0)

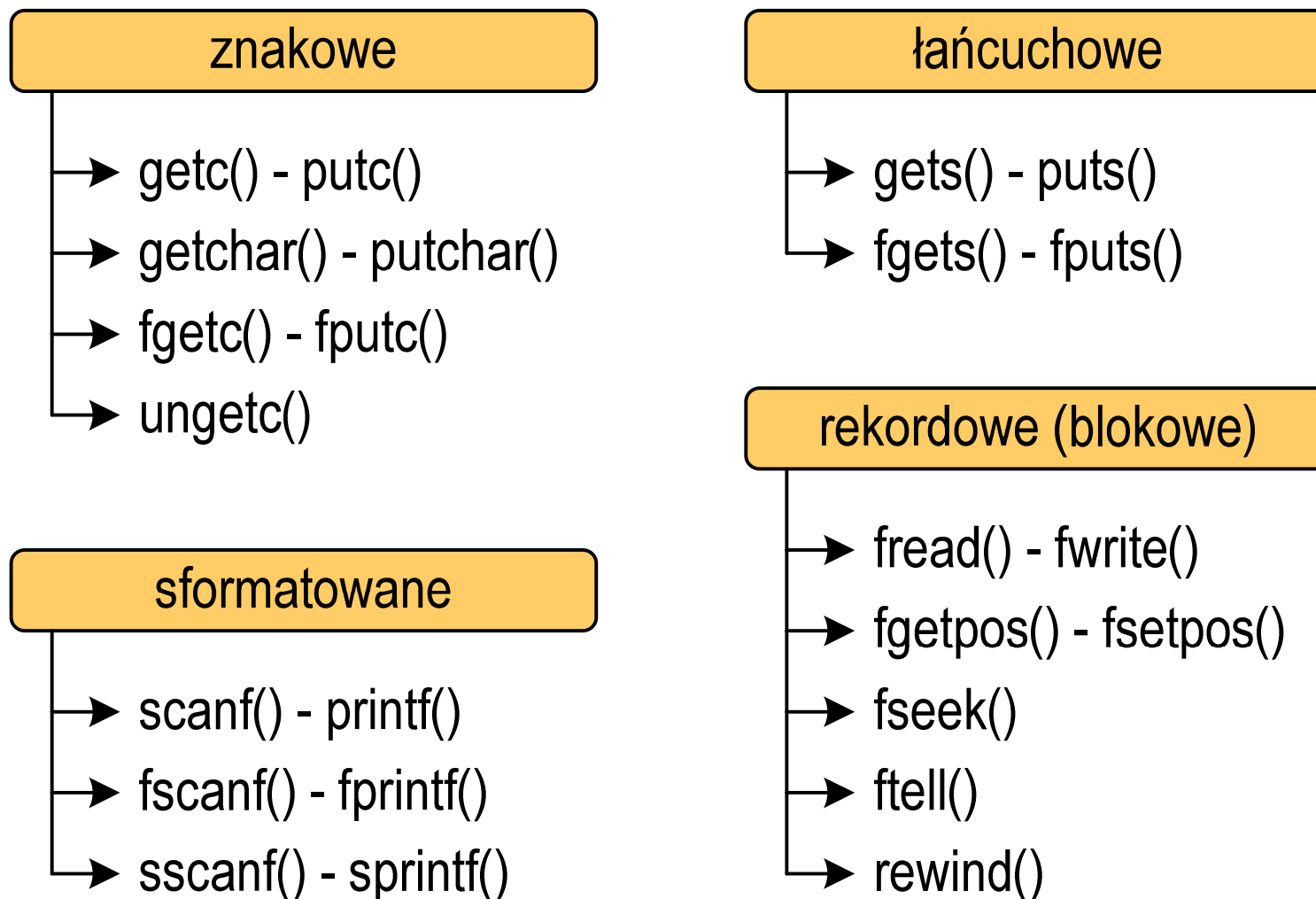


## Język C - stała znakowa

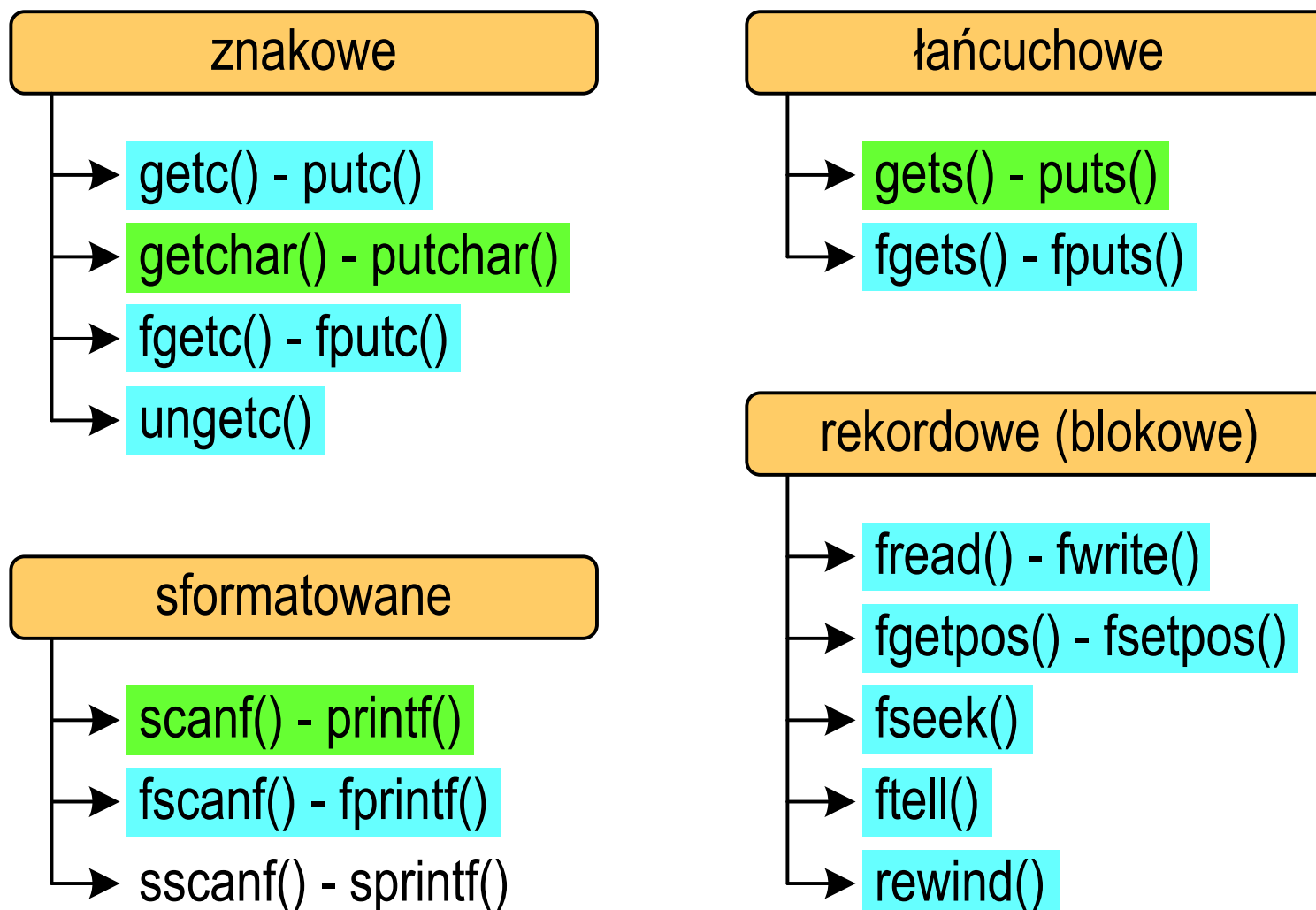
- Niektóre znaki mogą być reprezentowane w stałych znakowych przez sekwencje specjalne, które wyglądają jak dwa znaki, ale reprezentują tylko jeden znak

' \n ' - nowy wiersz	' \\ ' - \ (ang. backslash)
' \t ' - tabulator poziomy	' \' ' - apostrof
' \v ' - tabulator pionowy	' \" ' - cudzysłów
' \a ' - alarm	' \? ' - znak zapytania

# Język C - standardowe funkcje wejścia-wyjścia



# Język C - standardowe funkcje wejścia-wyjścia



## Język C - wyświetlenie tekstu

- Wyświetlenie tekstu funkcją `printf()` wymaga specyfikatora `%s`

```
char napis[15] = "Jan Kowalski";  
printf("Osoba: [%s]\n", napis);
```

```
Osoba: [Jan Kowalski]
```

- W specyfikatorze `%s`: szerokość określa szerokość pola, zaś precyzja - liczbę pierwszych znaków z łańcucha

```
char napis[15] = "Jan Kowalski";  
printf("[%10.6s]\n", napis);
```

```
[    Jan Ko]
```

## Język C - wyświetlenie tekstu

- Do wyświetlenia tekstu można zastosować funkcję `puts()`

```
puts()
```

```
int puts(const char *s);
```

- Funkcja `puts()` wypisuje na `stdout` (ekran) zawartość łańcucha znakowego (ciąg znaków zakończony znakiem `'\0'`), zastępując znak `'\0'` znakiem `'\n'`

```
char napis[15] = "Jan Kowalski";  
puts(napis);
```

```
Jan Kowalski
```

## Język C - wyświetlenie znaku

- Wyświetlenie znaku funkcją `printf()` wymaga specyfikatora `%c`

```
char zn = 'x';  
printf("Znak to: [%c]\n", zn);
```

```
Znak to: [x]
```

- Do wyświetlenia znaku można zastosować także funkcję `putchar()`

```
putchar()      int putchar(int znak);
```

```
putchar('K'); putchar(111); putchar(0x74);
```

```
Kot
```

## Język C - wyświetlenie tekstu

- Łańcuch znaków jest zwykłą tablicą - można więc odwoływać się do jej pojedynczych elementów

```
char txt[15] = "Ola ma laptopa";  
printf("Znaki: ");  
for (int i=0; i<15; i++) printf("%c ",txt[i]);
```

```
Znaki: O l a   m a   l a p t o p a
```

```
printf("Kody: ");  
for (int i=0; i<15; i++) printf("%d ",txt[i]);
```

```
Kody:  79 108 97 32 109 97 32 108 97 112 116 111 112 97 0
```

## Język C - wczytanie tekstu

- Do wczytania tekstu funkcją `scanf()` stosowany jest specyfikator `%s`

```
char napis[15];  
scanf("%s", napis);
```

brak znaku `&`



- W specyfikatorze formatu `%s` można podać szerokość

```
char napis[15];  
scanf("%10s", napis);
```

- W powyższym przykładzie `scanf()` zakończy wczytywanie tekstu po pierwszym białym znaku (spacja, tabulacja, enter) lub w momencie pobrania 10 znaków



## Język C - wczytanie tekstu

- W przypadku wprowadzenia tekstu "To jest napis", funkcja `scanf()` zapamięta tylko wyraz "To"
- Zapamiętanie całego wiersza tekstu (do naciśnięcia klawisza `Enter`) wymaga użycia funkcji `gets()`

```
gets ( )
```

```
char *gets (char *s) ;
```

- Funkcja `gets()` wprowadza wiersz (ciąg znaków zakończony `'\n'`) ze strumienia `stdin` (klawiatura) i umieszcza w obszarze pamięci wskazywanym przez wskaźnik `s` zastępując `'\n'` znakiem `'\0'`

```
char napis [15] ;  
gets (napis) ;
```

## Język C - wczytanie znaku

- Wczytanie jednego znaku funkcją `scanf()` wymaga specyfikatora formatu `%c` (przed zmienną `znak` musi wystąpić operator `&`)

```
int znak;  
scanf ("%c", &znak);
```

- Do wczytania znaku można zastosować także funkcję `getchar()`

<code>getchar()</code>	<code>int getchar(void);</code>
------------------------	---------------------------------

```
int znak;  
znak = getchar();
```

## Język C - plik nagłówkowy string.h

`strcpy()`

```
char *strcpy(char *s1, const char *s2);
```

- Kopiuje łańcuch `s2` do łańcucha `s1`

`strlen()`

```
size_t strlen(const char *s);
```

- Zwraca długość łańcucha znaków, nie uwzględnia znaku `'\0'`

`strcat()`

```
char *strcat(char *s1, const char *s2);
```

- Dołącza do łańcucha `s1` łańcuch `s2`

## Język C - plik nagłówkowy string.h

`strcmp()`

```
int strcmp(const char *s1, const char *s2);
```

- Porównuje łańcuchy `s1` i `s2` z rozróżnianiem wielkości liter

`strncmpi()`

```
int strncmpi(const char *s1, const char *s2);
```

- Porównuje łańcuchy `s1` i `s2` bez rozróżniania wielkości liter

`strchr()`

```
char *strchr(const char *s, int c);
```

- Szuka w łańcuchu `s` znaku `c`

## Język C - plik nagłówkowy string.h

`strlwr()`

`char *strlwr(char *s);`

- Zamienia w łańcuchu **s** wielkie litery na małe

`strupr()`

`char *strupr(char *s);`

- Zamienia w łańcuchu **s** małe litery na wielkie

`strrev()`

`char *strrev(char *s);`

- Odwraca kolejność znaków w łańcuchu **s**

## Język C - plik nagłówkowy string.h (przykład)

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char napis1[] = "Tekst w buforze", napis2[20];

    printf("napis1: %s \n", napis1);
    int dlugosc = strlen(napis1);
    printf("liczba znakow w napis1: %d \n", dlugosc);
    strcpy(napis2, napis1);
    printf("napis2: %s \n", napis2);
    strrev(napis2);
    printf("napis2 (odwr): %s \n", napis2);

    return 0;
}
```

## Język C - plik nagłówkowy string.h (przykład)

```
#include <stdio.h>
#include <string.h>
```

```
int main(void)
{
```

```
    char napis1[] = "T
```

```
    printf("napis1: %s \n", napis1);
```

```
    int dlugosc = strlen(napis1);
```

```
    printf("liczba znakow w napis1: %d \n", dlugosc);
```

```
    strcpy(napis2, napis1);
```

```
    printf("napis2: %s \n", napis2);
```

```
    strrev(napis2);
```

```
    printf("napis2 (odwr): %s \n", napis2);
```

```
    return 0;
```

```
}
```

```
napis1: Tekst w buforze
liczba znakow w napis1: 15
napis2: Tekst w buforze
napis2 (odwr): ezrofub w tskeT
```



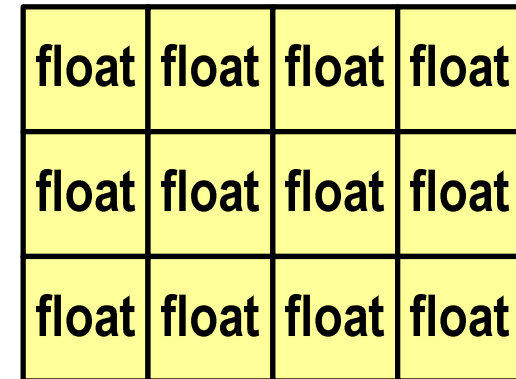
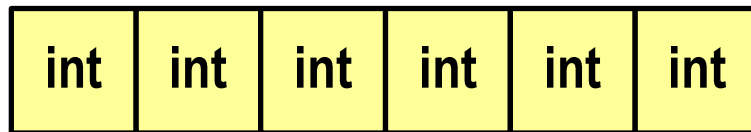




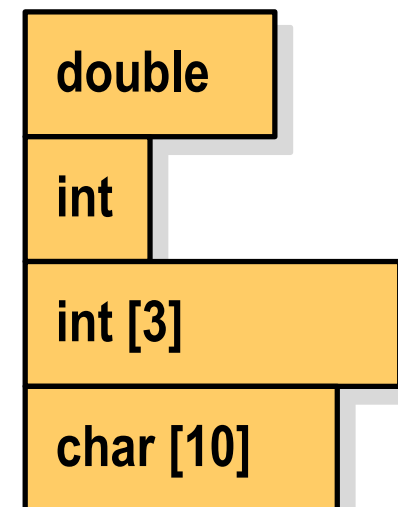


# Struktury w języku C

- **Tablica** - ciągły obszar pamięci zawierający elementy tego samego typu



- **Struktura** - zestaw elementów różnych typów, zgrupowanych pod jedną nazwą



## Deklaracja struktury

```
struct nazwa
{
    opis_pola_1;
    opis_pola_2;
    ...
    opis_pola_n;
};
```

```
struct punkt
{
    int x;
    int y;
};
```

- Elementy struktury to **pola** (dane, komponenty, składowe) struktury
- Deklaracje pól mają taką samą postać jak deklaracje zmiennych
- Deklarując strukturę tworzymy nowy typ danych (**struct punkt**), którym można posługiwać się tak samo jak każdym innym typem standardowym

## Deklaracja struktury

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

```
struct zesp
{
    float Re, Im;
};
```

- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola struktury)
- Zapisanie danych do struktury wymaga zdefiniowania **zmiennej strukturalnej**

# Deklaracja zmiennej strukturalnej

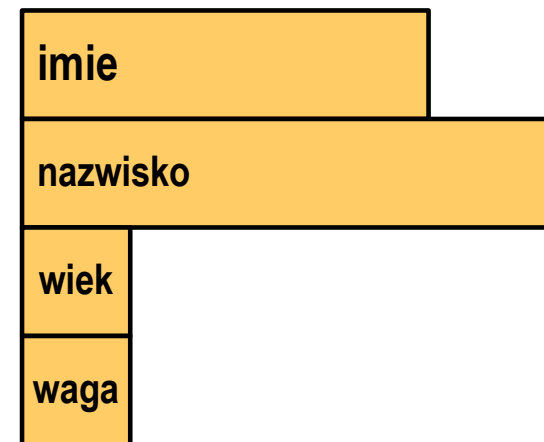
```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal ;

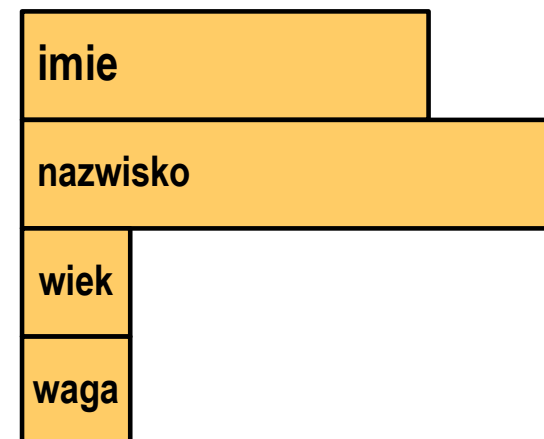
int main(void)
{
    struct osoba Nowak ;
    ...
}
```

- **Kowal, Nowak** - zmienne typu **struct osoba**

**Kowal**



**Nowak**



## Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

- Operator `.` nazywany jest **operatorem bezpośredniego wyboru pola**
- Zapisanie wartości do pól zmiennej **Nowak** ma postać

```
Nowak.wiek = 25;  
strcpy(Nowak.imie, "Jan");
```

- Wyrażenie **Nowak.wiek** traktowane jest jak zmienna typu **int**, zaś wyrażenie **Nowak.imie** traktowane jest jak łańcuch znaków

```
printf("%s - wiek %d\n", Nowak.imie, Nowak.wiek);  
scanf("%d", &Nowak.wiek);  
gets(Nowak.imie);
```

## Odwołania do pól struktury

- Gdy zmienna strukturalna jest wskaźnikiem, to do odwołania do pola struktury używamy **operatora pośredniego wyboru pola (->)**

```
wskaźnik_do_struktury -> nazwa_pola
```

```
struct osoba Nowak, *Nowak1;  
Nowak1 = &Nowak;  
Nowak1 -> wiek = 25;  
  
/* lub */  
  
(*Nowak1).wiek = 25;
```

- W ostatnim zapisie nawiasy są konieczne, gdyż operator **.** ma wyższy priorytet niż operator **\***



## Struktury - przykład (osoba)

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek;
};

int main(void)
{
    struct osoba Nowak;
```

## Struktury - przykład (osoba)

```
printf("Imie:      ");  
gets(Nowak.imie);  
  
printf("Nazwisko: ");  
gets(Nowak.nazwisko);  
  
printf("Wiek:      ");  
scanf("%d", &Nowak.wiek);  
  
printf("%s %s, wiek: %d\n", Nowak.imie,  
      Nowak.nazwisko, Nowak.wiek);  
  
return 0;  
}
```

```
Imie:      Jan  
Nazwisko:  Nowak  
Wiek:      22  
Jan Nowak, wiek: 22
```

## Struktury w języku C

- **Inicjalizacja** może dotyczyć tylko zmiennych strukturalnych, nie można inicjalizować pól w deklaracji struktury

```
struct osoba
{
    char imie[15], nazwisko[20];
    int wiek, waga;
};
```

```
struct osoba Nowak = {"Jan", "Nowak", 25, 74};
```

- Do zmiennych strukturalnych można stosować **operator =**

```
struct osoba Kowal = {"Ewa", "Kowal", 21, 54};
struct osoba Kowal1;
Kowal1 = Kowal;
```

## Struktury - przykład (miernik)

```
#include <stdio.h>

struct miernik
{
    double k;    // klasa dokładności
    int d;      // liczba działek podziałki
    double Zp;  // zakres pomiarowy
};

int main(void)
{
    // Amperomierz LE-3P
    struct miernik LE3P = {0.5, 60, 12};
    double Dpm, p;
```



## Struktury - przykład (miernik)

```
printf("Amperomierz analogowy LE-3P\n");  
printf("Zakres pomiarowy: %g A\n", LE3P.Zp);  
printf("Liczba dzialek podzialki: %d\n", LE3P.d);  
printf("Klasa dokladnosci: %g\n", LE3P.k);  
printf("-----\n");  
  
printf("Bezwzględny maksymalny bład pomiaru:\n");  
p = 0.2;  
Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);  
printf("* dla p = %g, Dpm = %g A\n", p, Dpm);  
  
p = 0.5;  
Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);  
printf("* dla p = %g, Dpm = %g A\n", p, Dpm);  
  
return 0;  
}
```

## Struktury - przykład (miernik)

```
printf("Amperomi  
printf("Zakres p  
printf("Liczba d  
printf("Klasa do  
printf("-----  
printf("Bezwzgle  
p = 0.2;  
Dpm = LE3P.Zp*(L  
printf("* dla p = %g, Dpm = %g A\n", p, Dpm);  
  
p = 0.5;  
Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);  
printf("* dla p = %g, Dpm = %g A\n", p, Dpm);  
return 0;  
}
```

```
Amperomierz analogowy LE-3P  
Zakres pomiarowy: 12 A  
Liczba dzialek podzialki: 60  
Klasa dokladnosci: 0.5  
-----  
Bezwzgly maksymalny blad pomiaru:  
* dla p = 0.2, Dpm = 0.1 A  
* dla p = 0.5, Dpm = 0.16 A
```

## Złożone deklaracje struktur

```
struct punkt  
{  
    int x;  
    int y;  
} tab[3];
```

tab

0	x	y
1	x	y
2	x	y

```
tab[0].x = 10;  
tab[0].y = 20;  
tab[1].x = 15;  
...
```

```
struct trojkat  
{  
    int nr;  
    struct punkt A, B, C;  
} Tr1;
```

Tr1

nr		
A	x	y
B	x	y
C	x	y

```
Tr1.nr = 1;  
Tr1.A.x = 10;  
Tr1.A.y = 20;  
Tr1.B.x = 15;  
...
```

Koniec wykładu nr 9

Dziękuję za uwagę!