Programowanie C (CP1S01005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. I, studia stacjonarne I stopnia

Rok akademicki 2024/2025

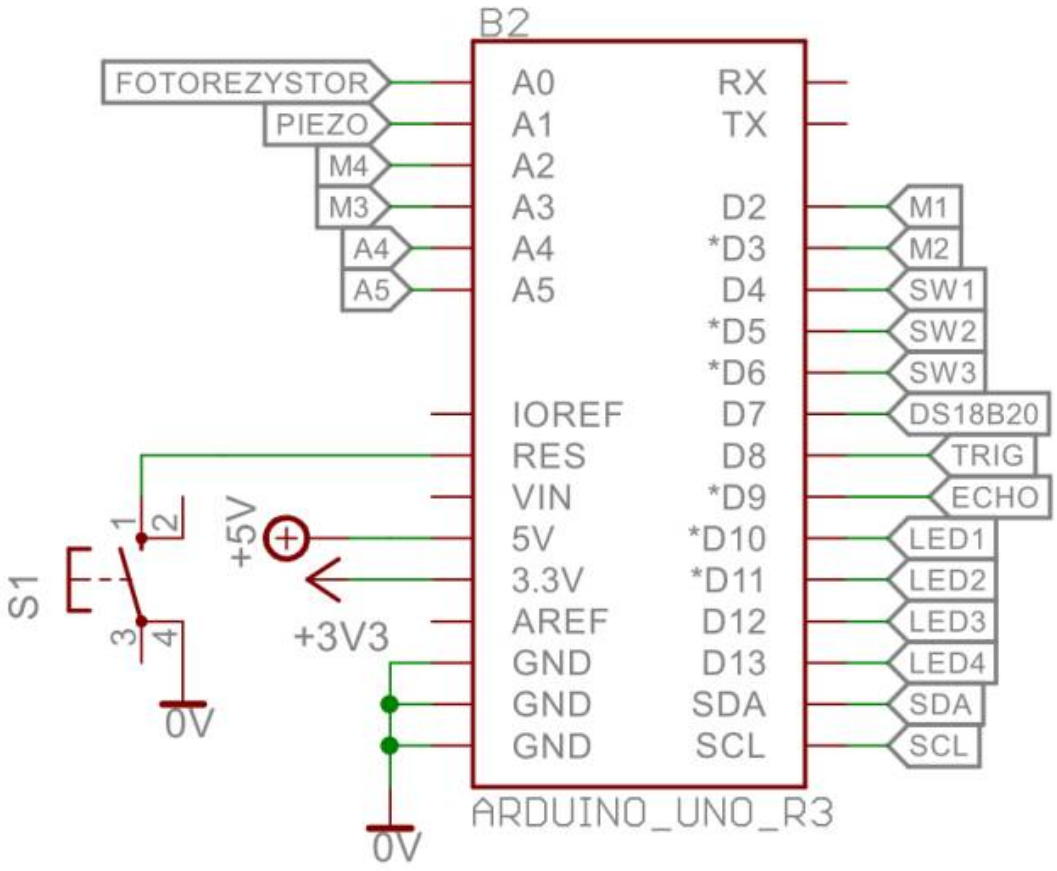
Wykład nr 7 (16.01.2025)

dr inż. Jarosław Forenc

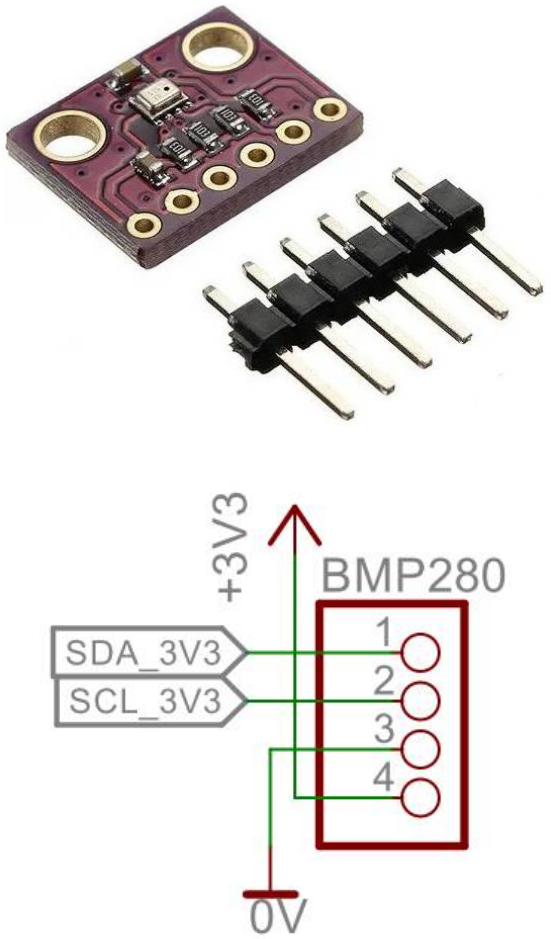
Plan wykładu nr 7

- Arduino
  - czujnik ciśnienia (BMP280)
  - wyświetlacz OLED
- Język C
  - łańcuchy znaków
  - struktury, pola bitowe, unie
  - operatory bitowe

Arduino - czujnik ciśnienia (BMP280)



Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
czujnika BMP280

Arduino - czujnik ciśnienia (BMP280)

- Specyfikacja:
  - napięcie zasilania: 3,3 V
  - zakres pomiarowy: od 300 hPa do 1100 hPa, dokładność: 1 hPa
  - interfejs komunikacyjny: I2C lub SPI
  - wbudowany termometr
  - pomiar temperatury w zakresie od -40 °C do +85 °C, dokładność: 1 °C
- Skompilowanie programu w Visual Studio Code wymaga dodania do projektu biblioteki Adafruit BMP280 Library by Adafruit
- Uruchomienie programu: PlatformIO → PROJECT TASK → Default → General → Upload and Monitor

Arduino - czujnik ciśnienia (BMP280)

Czujnik ciśnienia BMP280

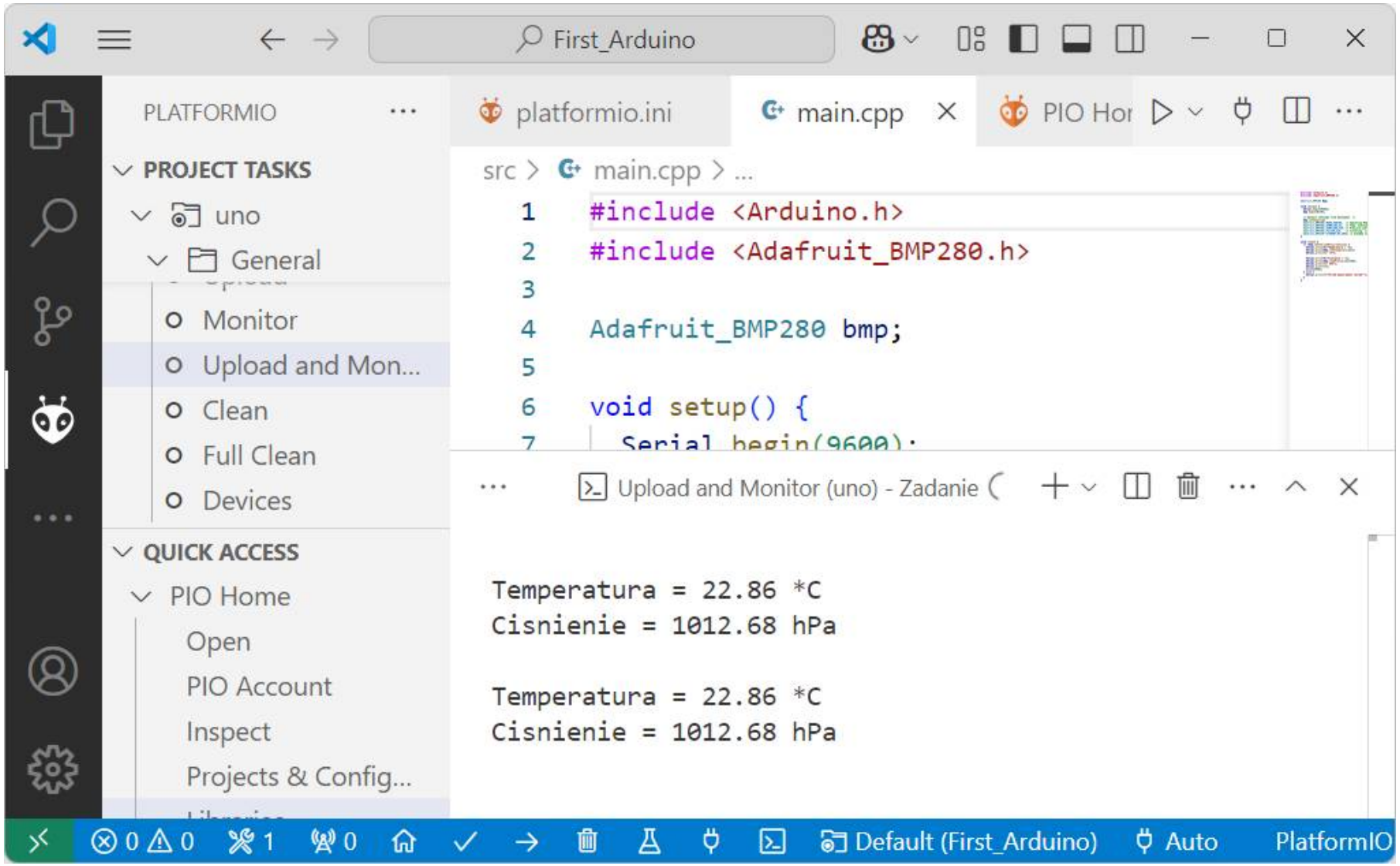
```
#include <Arduino.h>
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bmp;

void setup() {
  Serial.begin(9600);
  bmp.begin(0x76);

  /* Default settings from datasheet. */
  bmp.setSampling(
  Adafruit_BMP280::MODE_FORCED,  /* Operating Mode. */
  Adafruit_BMP280::SAMPLING_X2,  /* Temp. oversampling */
  Adafruit_BMP280::SAMPLING_X16, /* Pressure oversampling */
  Adafruit_BMP280::FILTER_X16,   /* Filtering. */
  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */
}
```

Arduino - czujnik ciśnienia (BMP280)

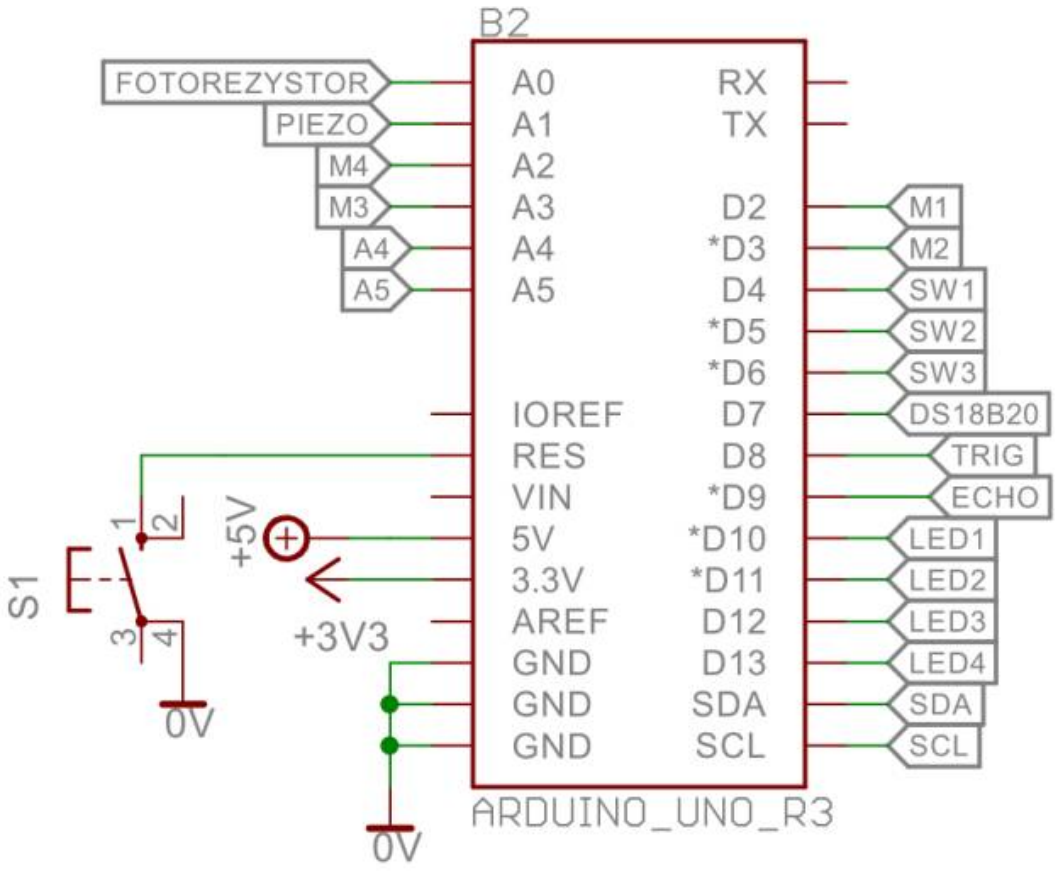
Czujnik ciśnienia BMP280

```
void loop() {
  if (bmp.takeForcedMeasurement()) {
    Serial.print(F("Temperatura = "));
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");

    Serial.print(F("Cisnienie = "));
    Serial.print(bmp.readPressure()/100);
    Serial.println(" hPa");
    Serial.println();
    delay(2000);
  } else {
    Serial.println("Forced measurement failed!");
  }
}
```

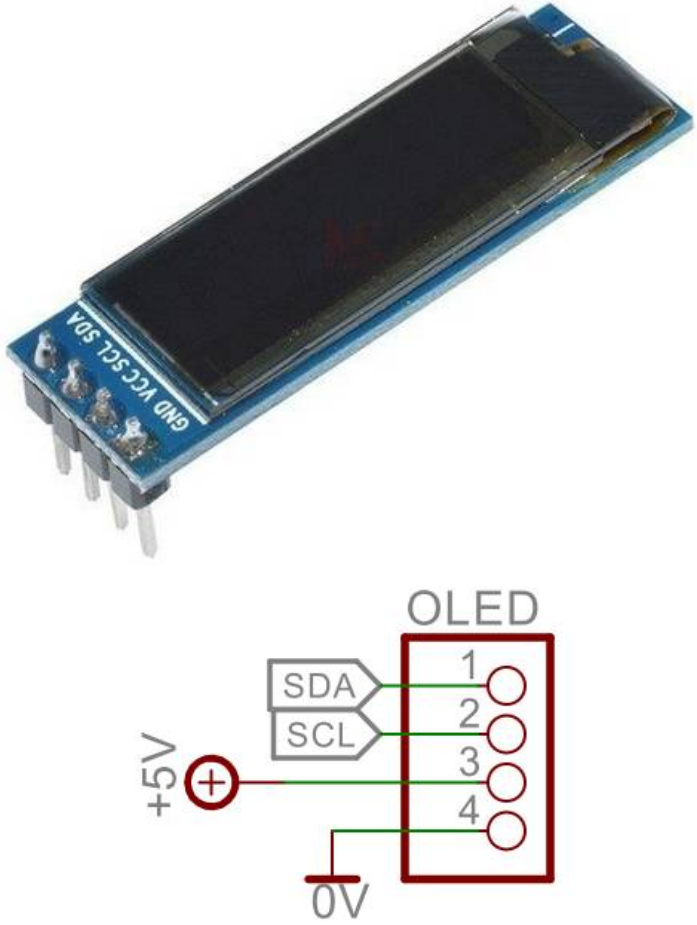
Arduino - czujnik ciśnienia (BMP280)



Arduino - wyświetlacz OLED 0,91"



Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
wyświetlacza OLED

Arduino - wyświetlacz OLED 0,91"

- Specyfikacja:
  - wyświetlacz OLED 0.91" o rozdzielczości 128 × 32
  - do zastosowań w systemach AVR, Arduino, PIC, STM32
  - komunikacja z wyświetlaczem przez interfejs I2C
  - kąt widzenia: ok. 160 stopni
  - układ sterownika IC: SSD1306
  - napięcia zasilania: 3,3 V lub 5 V DC
- Skompilowanie programu w Visual Studio Code wymaga dodania do projektu biblioteki Adafruit SSD1306 by Adafruit

Arduino - wyświetlacz OLED 0,91"

Wyświetlacz OLED 0,91"

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define SW1_PIN 4

#define OLED_RESET     -1
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
                         &Wire, OLED_RESET);
int counter = 0;
```

Arduino - wyświetlacz OLED 0,91"

Wyświetlacz OLED 0,91"

```
void setup() {
  Serial.begin(9600);
  Serial.println("OLED Start");
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  // Clear the buffer.
  display.clearDisplay();
  display.display();
  delay(1000);

  pinMode(SW1_PIN,INPUT);

  // text display tests
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0,0);
```

Arduino - wyświetlacz OLED 0,91"

```
  display.print("SW1: ");
  display.print(counter);
  display.display();
}
void loop() {
  if(digitalRead(SW1_PIN) == LOW)
  {
    delay(200);
    counter++;
    display.setCursor(0,0);
    display.clearDisplay();
    display.print("SW1: ");
    display.print(counter);
    display.display();
  }
}
```

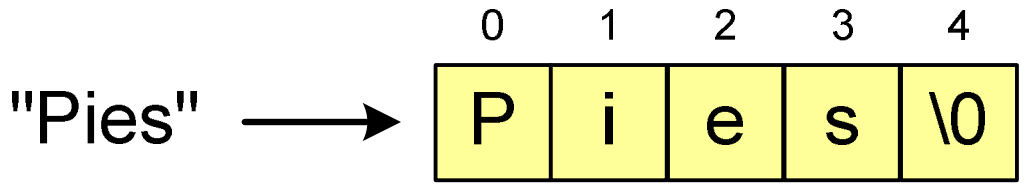
Wyświetlacz OLED 0,91"

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"
```

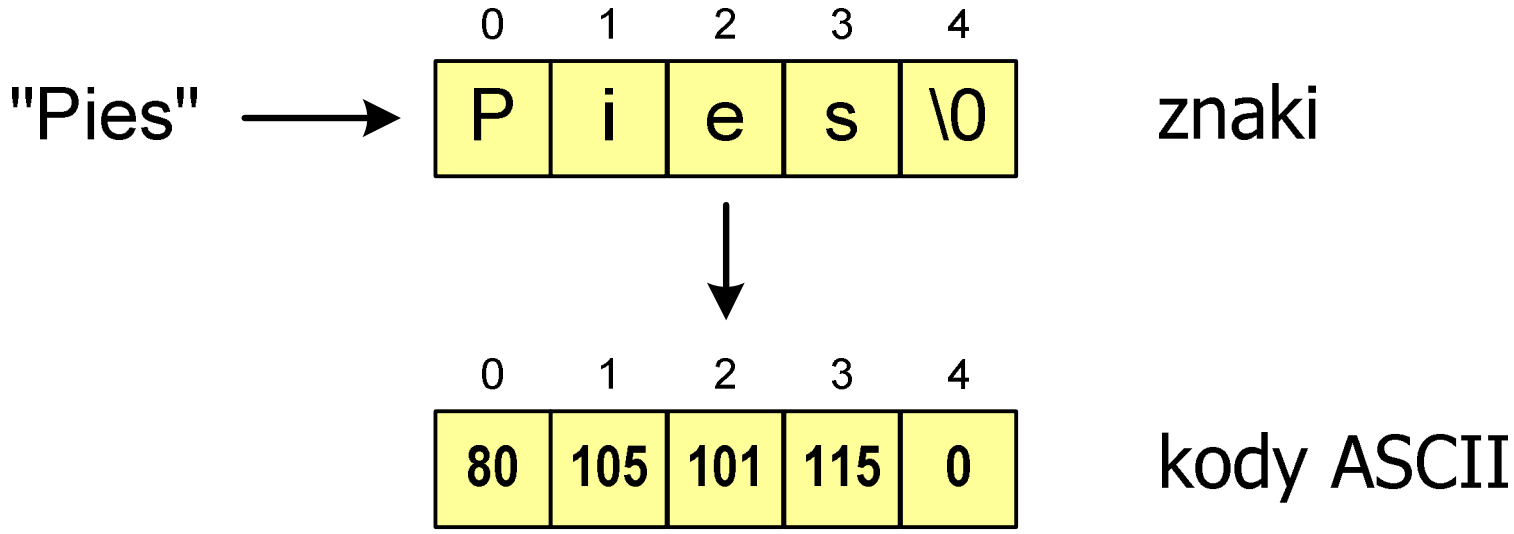
- Implementacja - tablica, której elementami są pojedyncze znaki (typ char)



- 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 wartości liczbowe)



Język C - deklaracja łańcucha znaków

- Deklaracja zmiennej przechowującej łańcuch znaków

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

Przykład:

```
chat 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;
```

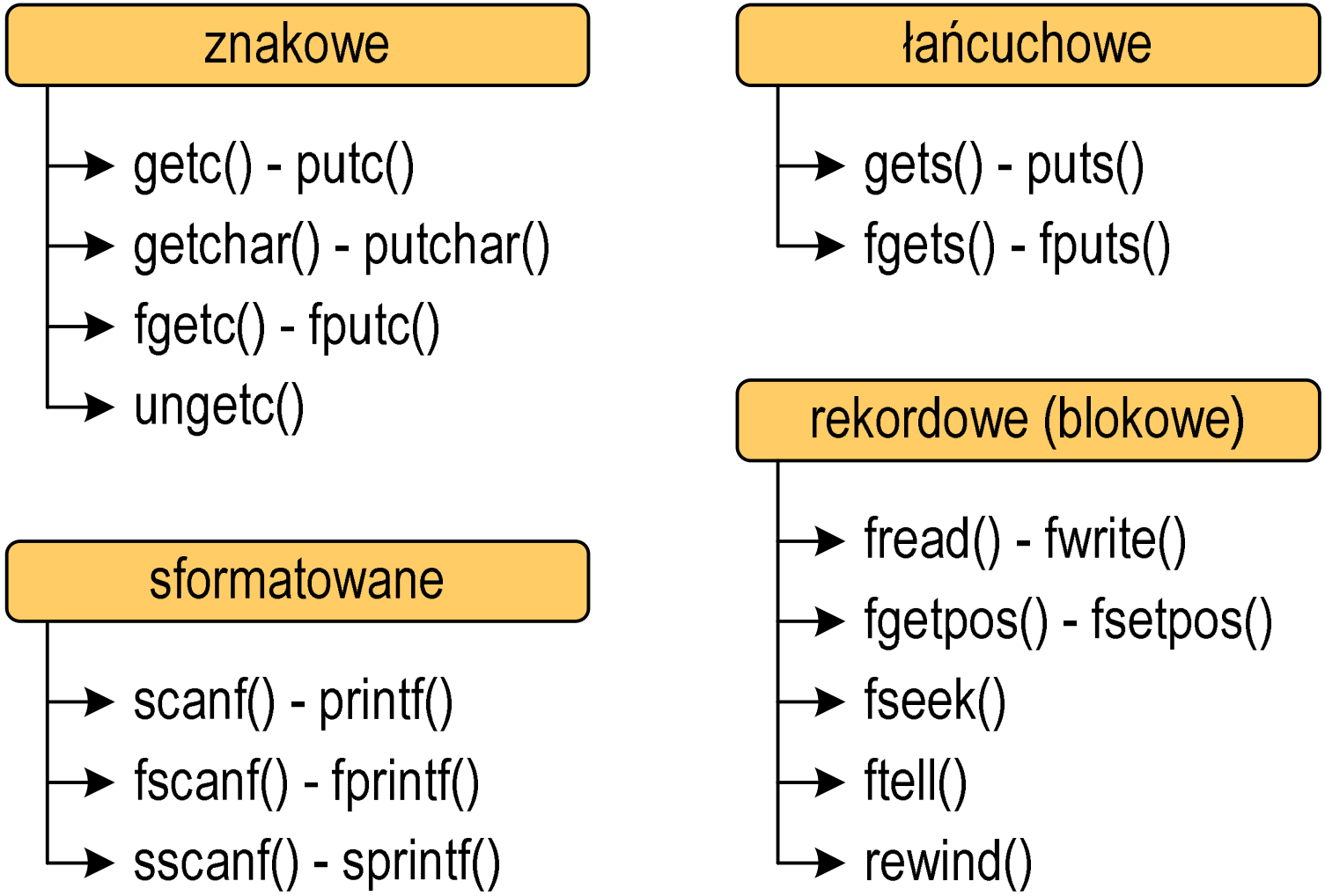
- Uwaga:
  - `'x'` - stała znakowa (jeden znak)
  - `"x"` - łańcuch znaków (dwa znaki: x oraz \0)

Język C - stała znakowa

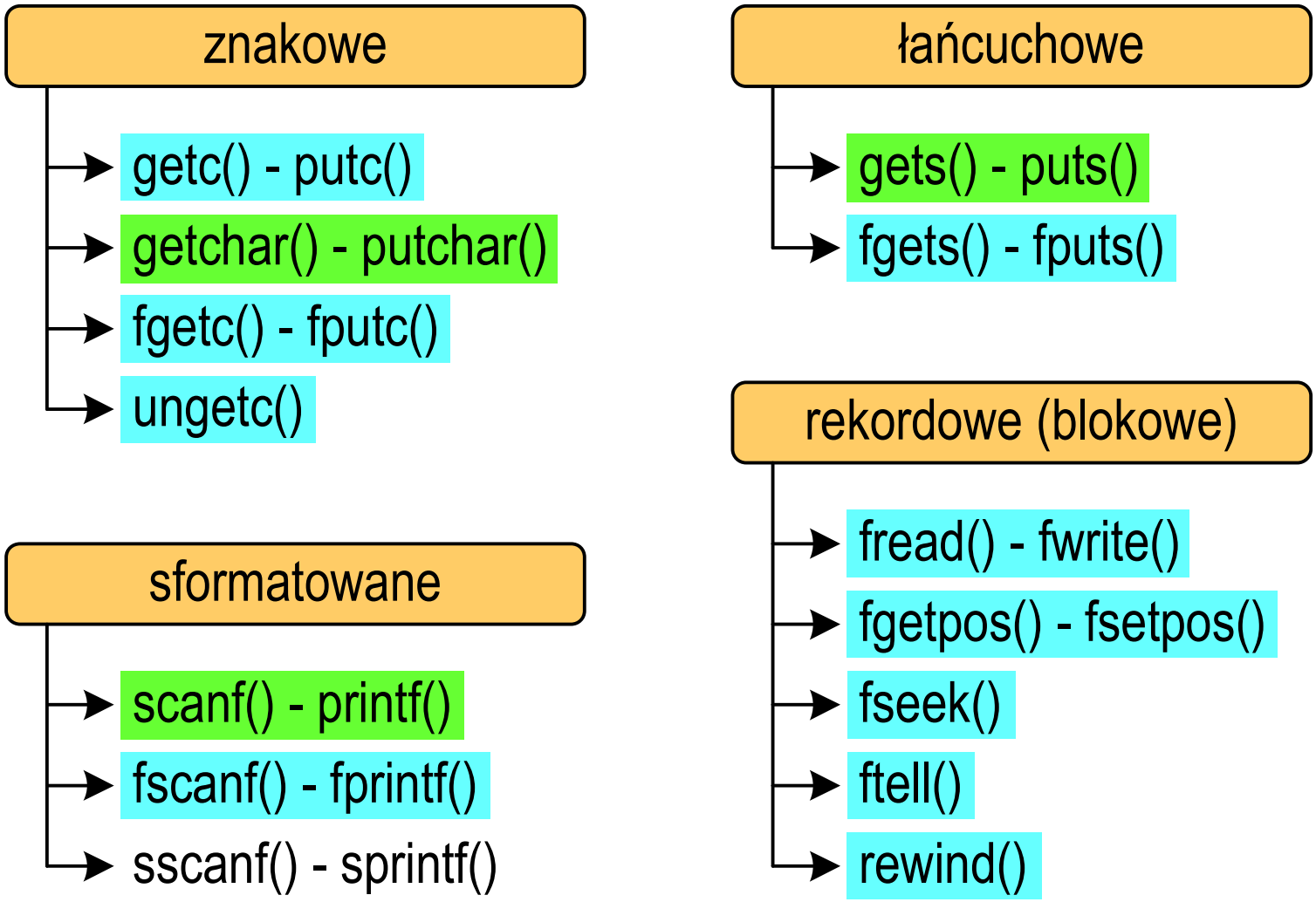
- Niektóre znaki mogą być reprezentowane w stałych znakowych przez sekwencje specjalne, które wyglądają jak dwa znaki, ale reprezentują tylko jeden znak

`'\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 znaku

- 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()

getchar()	int getchar(void);

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

Język C - macierz elementów typu char

- Szczególny przypadek tablicy dwuwymiarowej

```
char txt[3][15] = {"Programowanie",
                   "nie jest","trudne"};
```

- Tablica w pamięci komputera

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	P	r	o	g	r	a	m	o	w	a	n	i	e	\0	\0
1	n	i	e		j	e	s	t	\0	\0	\0	\0	\0	\0	\0
2	t	r	u	d	n	e	\0	\0	\0	\0	\0	\0	\0	\0	\0

Język C - macierz elementów typu char

- Używając dwóch indeksów (nr wiersza i nr kolumny) można odwoływać się do jej pojedynczych elementów (znaków)

```
char txt[3][15] = {"Programowanie",
                   "nie jest","trudne"};
for (int i=0; i<3; i++)
{
    for (int j=0; j<6; j++)
       printf("%c",txt[i][j]);
    printf("\n");
}
```

```
Progra
nie je
trudne
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	P	r	o	g	r	a	m	o	w	a	n	i	e	\0	\0
1	n	i	e		j	e	s	t	\0	\0	\0	\0	\0	\0	\0
2	t	r	u	d	n	e	\0	\0	\0	\0	\0	\0	\0	\0	\0

Język C - macierz elementów typu char

- Użycie jednego indeksu (numeru wiersza) powoduje potraktowanie całego wiersza jako łańcuch znaków (napisu)

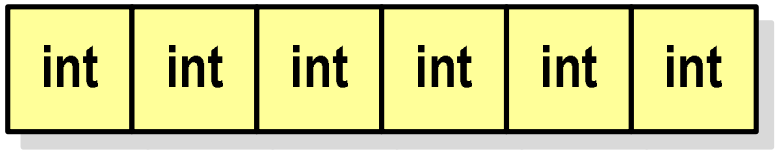
```
char txt[3][15] = {"Programowanie",
                   "nie jest","trudne"};
printf("%s ",txt[1]);
printf("%s ",txt[2]);
printf("%s ",txt[0]);
```

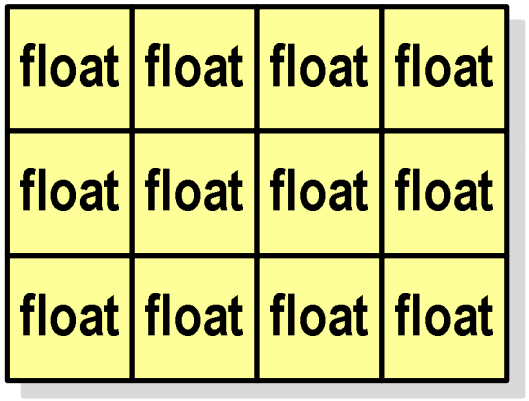
```
nie jest trudne Programowanie
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	P	r	o	g	r	a	m	o	w	a	n	i	e	\0	\0
1	n	i	e		j	e	s	t	\0	\0	\0	\0	\0	\0	\0
2	t	r	u	d	n	e	\0	\0	\0	\0	\0	\0	\0	\0	\0

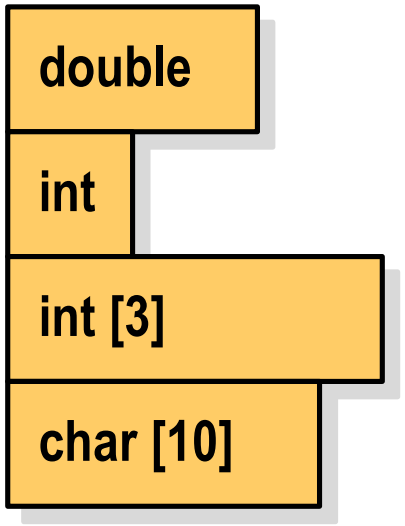
Struktury w języku C

- 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

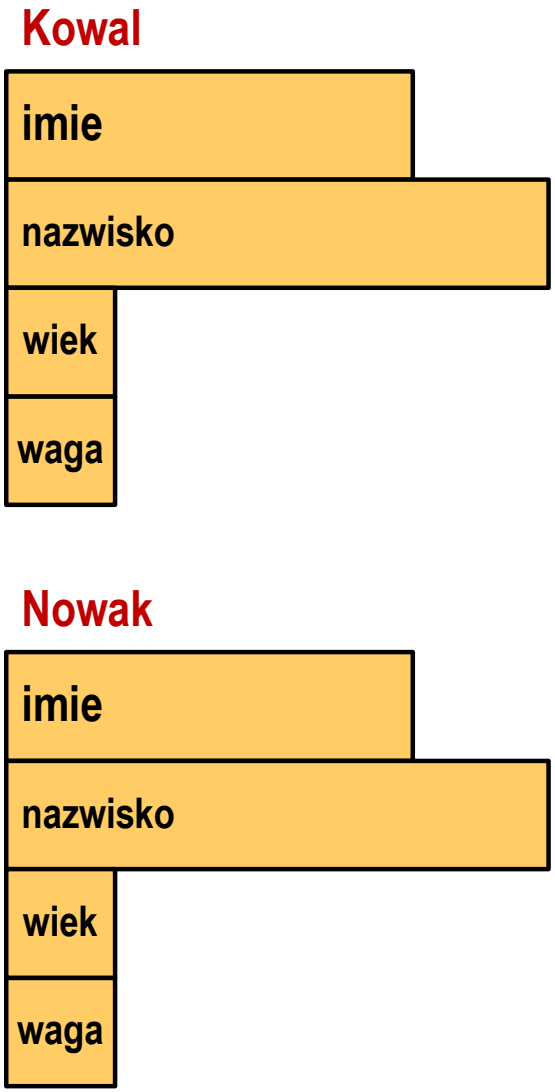
```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

```
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

Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_zmiennej_strukturalnej.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);
```

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 w języku C - przykład (data)

```c
#include <stdio.h>

struct date
{
    int day;
    int month;
    int year;
} day1;

int main(void)
{
    struct date day2 = {19,11,2018};
```

day1

day	?
month	?
year	?

day2

day	19
month	11
year	2018

Struktury w języku C - przykład (data)

```
    day1.day = 1;
    day1.month = 9;
    day1.year = 2018;

    printf("Date1: %02d-%02d-%4d\n",
       day1.day,day1.month,day1.year);
    printf("Date2: %02d-%02d-%4d\n",
       day2.day,day2.month,day2.year);

    return 0;
}
```

```
Date1: 01-09-2018
Date2: 19-11-2018
```

day1

day	1
month	9
year	2018

day2

day	19
month	11
year	2018

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("Bezwzgledny maksymalny blad 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;
}
```

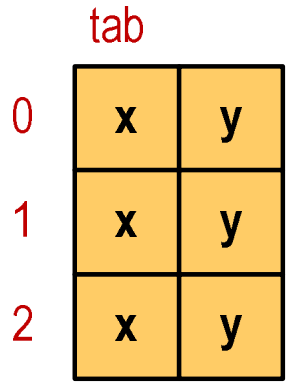
Struktury - przykład (miernik)

```
    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
-----------------------------------
Bezwzgledny maksymalny blad pomiaru:
* dla p = 0.2, Dpm = 0.1 A
* dla p = 0.5, Dpm = 0.16 A
```

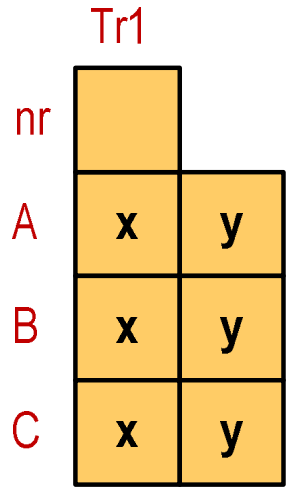
Złożone deklaracje struktur

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




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

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

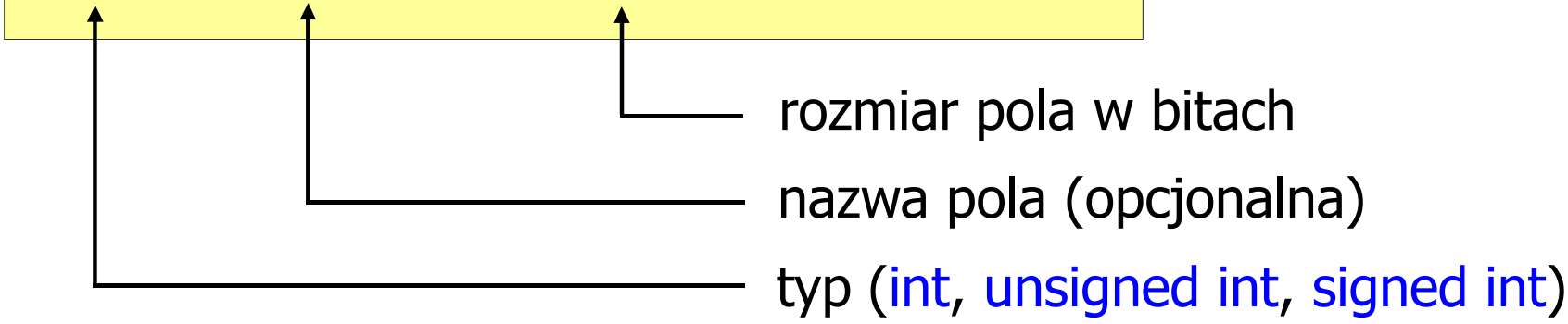



```
Tr1.nr = 1;
Tr1.A.x = 10;
Tr1.A.y = 20;
Tr1.B.x = 15;
…
```

Pola bitowe

- Umożliwiają dostęp do pojedynczych bitów oraz przechowywanie małych wartości zajmujących pojedyncze bity
- Pola bitowe deklarowane są wewnątrz struktur

```
typ id_pola : wielkość_pola;
```



- Wartości zapisane w polach traktowane są jak liczby całkowite
- Zakres wartości pól wynika z wielkości_pola

Pola bitowe

```
struct Bits
{
    unsigned int a : 4;     /* zakres: 0...15 */
    unsigned int b : 2;     /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;     /* zakres: 0...63 */
};
```

- Dostęp do pól bitowych odbywa się na takiej samej zasadzie jak do normalnych pól struktury

```
struct Bits dane;

dane.a = 10;
dane.b = 3;
```

Pola bitowe

```
struct Bits
{
    unsigned int a : 4;     /* zakres: 0...15 */
    unsigned int b : 2;     /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;     /* zakres: 0...63 */
};
```

- Jeśli pole nie ma nazwy, to nie można się do niego odwoływać
- Pola bitowe nie mają adresów
  - nie można wobec pola bitowego stosować operatora & (adres)
  - nie można polu bitowemu nadać wartości funkcją scanf()

Pola bitowe - przykład

```
struct Flags_8086
{
    unsigned int CF : 1;    /* Carry Flag */
    unsigned int    : 1;
    unsigned int PF : 1;    /* Parity Flag */
    unsigned int    : 1;
    unsigned int AF : 1;    /* Auxiliary - Carry Flag */
    unsigned int    : 1;
    unsigned int ZF : 1;    /* Zero Flag */
    unsigned int SF : 1;    /* Signum Flag */
    unsigned int TF : 1;    /* Trap Flag */
    unsigned int IF : 1;    /* Interrupt Flag */
    unsigned int DF : 1;    /* Direction Flag */
    unsigned int OF : 1;    /* Overflow Flag */
};
```

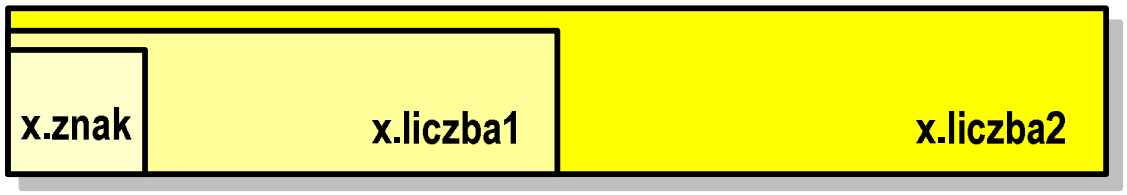
Unie

- Specjalny rodzaj struktury umożliwiający przechowywanie danych różnych typów w tym samym obszarze pamięci
- Do przechowywania wartości w unii należy zadeklarować zmienną

```
union zbior
{
    char   znak;
    int    liczba1;
    double liczba2;
};
```

```
union zbior x;
```

- Zmienna x może przechowywać wartość typu char lub typu int lub typu double, ale tylko jedną z nich w danym momencie



- Rozmiar unii wyznaczany jest przez rozmiar największego jej pola

Unie

- Dostęp do pól unii jest taki sam jak do pól struktury

```
union zbior x;

x.znak = 'a';
x.liczba2 = 12.15;
```

```
union zbior
{
    char   znak;
    int    liczba1;
    double liczba2;
};
```

- Unię można zainicjować jedynie wartością o typie jej pierwszej składowej
- Unie tego samego typu można sobie przypisywać

```
union zbior x = {'a'};
union zbior y;

y = x;
```

Język C - operatory bitowe

- Operatory bitowe wykonują operacje na poszczególnych bitach liczb
- Operatory bitowe można stosować jedynie do argumentów całkowitych typu char, short, int, long (ze znakiem lub bez)

Operator	Znaczenie	Opis
`&`	**AND**	dwuargumentowy operator koniunkcji bitowej
`|`	**OR**	dwuargumentowy operator alternatywy bitowej
`~`	**NOT**	jednoargumentowy operator uzupełnienia jedynkowego (zastępuje 0 → 1, 1 → 0)

Język C - operatory bitowe

- Operatory bitowe wykonują operacje na poszczególnych bitach liczb
- Operatory bitowe można stosować jedynie do argumentów całkowitych typu char, short, int, long (ze znakiem lub bez)

Operator	Znaczenie	Opis
^	XOR	dwuargumentowy operator różnicy symetrycznej
>>		dwuargumentowy operator przesunięcia bitowego w prawo
<<		dwuargumentowy operator przesunięcia bitowego w lewo

Język C - operator koniunkcji bitowej (&)

- Operator koniunkcji bitowej (&, AND) ustawia jedynkę na każdej pozycji bitowej tam, gdzie oba bity są równe jeden
- W pozostałych przypadkach ustawia zero

```
unsigned char x = 106;  /* 01101010 */
unsigned char y = 173;  /* 10101101 */
unsigned char z;
z = x & y;
```

```
x → 0 1 1 0 1 0 1 0
y → 1 0 1 0 1 1 0 1
---------------------
z → 0 0 1 0 1 0 0 0
```

x	0	1	0	1
y	0	0	1	1
x & y	0	0	0	1

Język C - operator alternatywy bitowej (|)

- Operator alternatywy bitowej (|, OR) ustawia jedynkę na tej pozycji bitowej, na której przynajmniej jeden z bitów jest równy jeden
- Ustawia zero, gdy oba bity są równe zero

```
unsigned char x = 106;  /* 01101010 */
unsigned char y = 173;  /* 10101101 */
unsigned char z;
z = x | y;
```

```
x → 0 1 1 0 1 0 1 0
y → 1 0 1 0 1 1 0 1
---------------------
z → 1 1 1 0 1 1 1 1
```

x	0	1	0	1
y	0	0	1	1
x | y	0	1	1	1

Język C - operator uzupełnienia jedynkowego (~)

- Operator uzupełnienia jedynkowego czyli negacji (~, NOT) zastępuje jedynkę zerem i zero jedynką

```
unsigned char x = 106;  /* 01101010 */
unsigned char z;
z = ~x;
```

```
 x → 0 1 1 0 1 0 1 0
---------------------
 z → 1 0 0 1 0 1 0 1
```

x	0	1
~x	1	0

Język C - operator różnicy symetrycznej (^)

- Operator różnicy symetrycznej (^, XOR) ustawia jedynkę na każdej pozycji bitowej tam, gdzie bity są różne, a zero tam, gdzie bity są takie same

```
unsigned char x = 106;  /* 01101010 */
unsigned char y = 173;  /* 10101101 */
unsigned char z;
z = x ^ y;
```

```
x → 0 1 1 0 1 0 1 0
y → 1 0 1 0 1 1 0 1
---------------------
z → 1 1 0 0 0 1 1 1
```

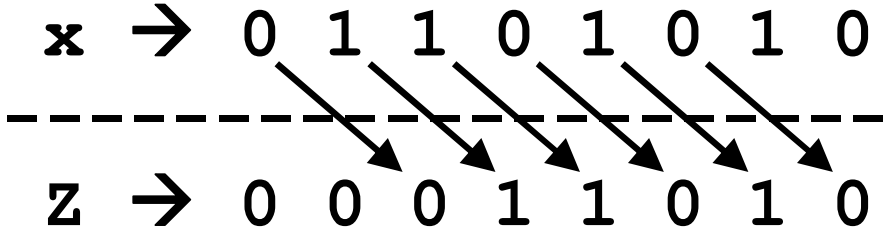
x	0	1	0	1
y	0	0	1	1
x ^ y	0	1	1	0

Język C - operator przesunięcia bitowego (>>)

- Operatory przesunięcia bitowego w prawo (>>) przesuwa bity argumentu stojącego po lewej stronie operatora o liczbę pozycji określoną przez argument stojący po prawej stronie operatora
- Drugi argument musi być liczbą dodatnią

```
unsigned char x = 106;  /* 01101010 */
unsigned char z;
z = x >> 2;
```

- Przesunięcie w prawo powoduje pojawienie się na najstarszej pozycji 0 (dla liczb bez znaku) lub powielenie bitu znaku (dla liczb ze znakiem)

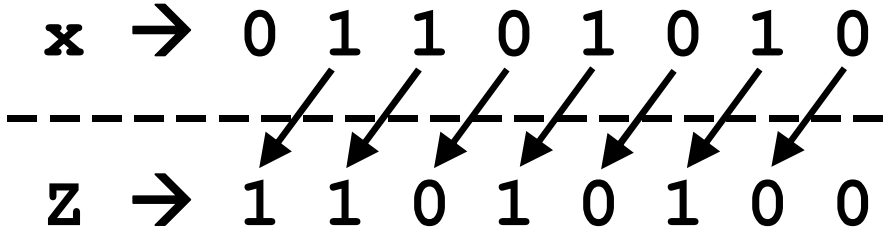


Język C - operator przesunięcia bitowego (<<)

- Operatory przesunięcia bitowego w lewo (<<) przesuwa bity argumentu stojącego po lewej stronie operatora o liczbę pozycji określoną przez argument stojący po prawej stronie operatora
- Drugi argument musi być liczbą dodatnią

```
unsigned char x = 106;  /* 01101010 */
unsigned char z;
z = x << 1;
```

- Przy przesunięciu w lewo zwolnione (najmłodsze) bity zapełniane są 0



Koniec wykładu nr 7

Dziękuję za uwagę!