# Programowanie mikrokontrolerów w języku wysokiego poziomu 1

(TS1F1008)

Politechnika Białostocka - Wydział Elektryczny
Elektronika i telekomunikacja, sem. I, studia stacjonarne I stopnia
Rok akademicki 2024/2025

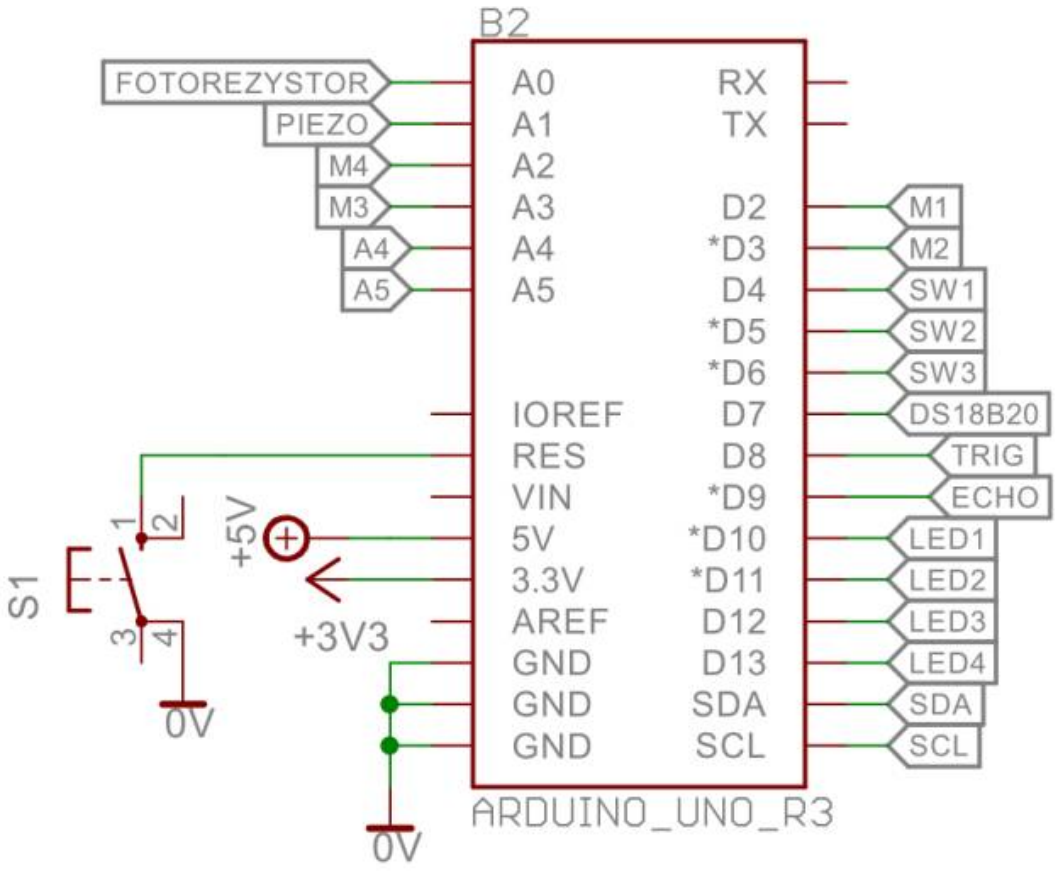
## Wykład nr 7 (23.01.2025)

dr inż. Jarosław Forenc

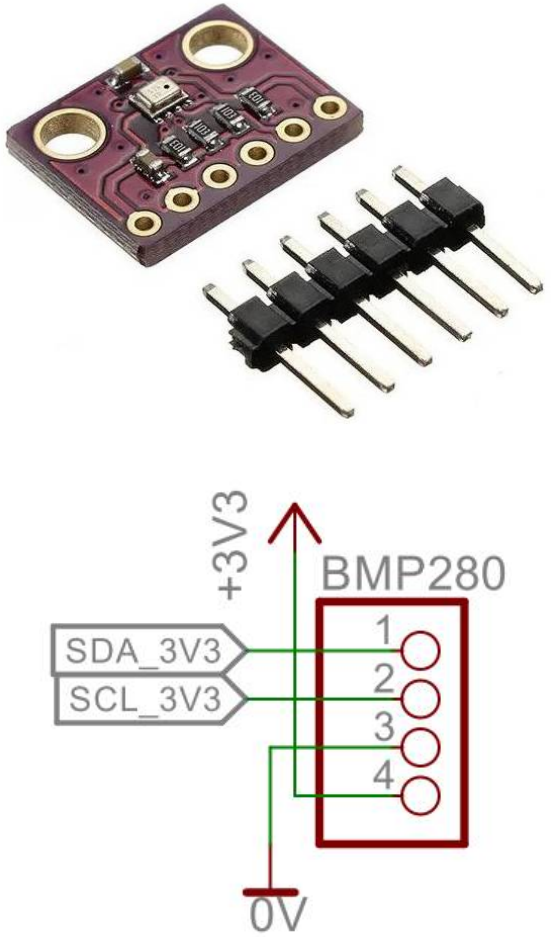
# Plan wykładu nr 7

- Arduino
  - czujnik ciśnienia (BMP280)
  - wyświetlacz OLED
- Język C
  - operatory bitowe
  - wskaźniki - deklaracja, przypisanie wartości
  - związek wskaźników z tablicami i strukturami
  - operacje na wskaźnikach

# 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

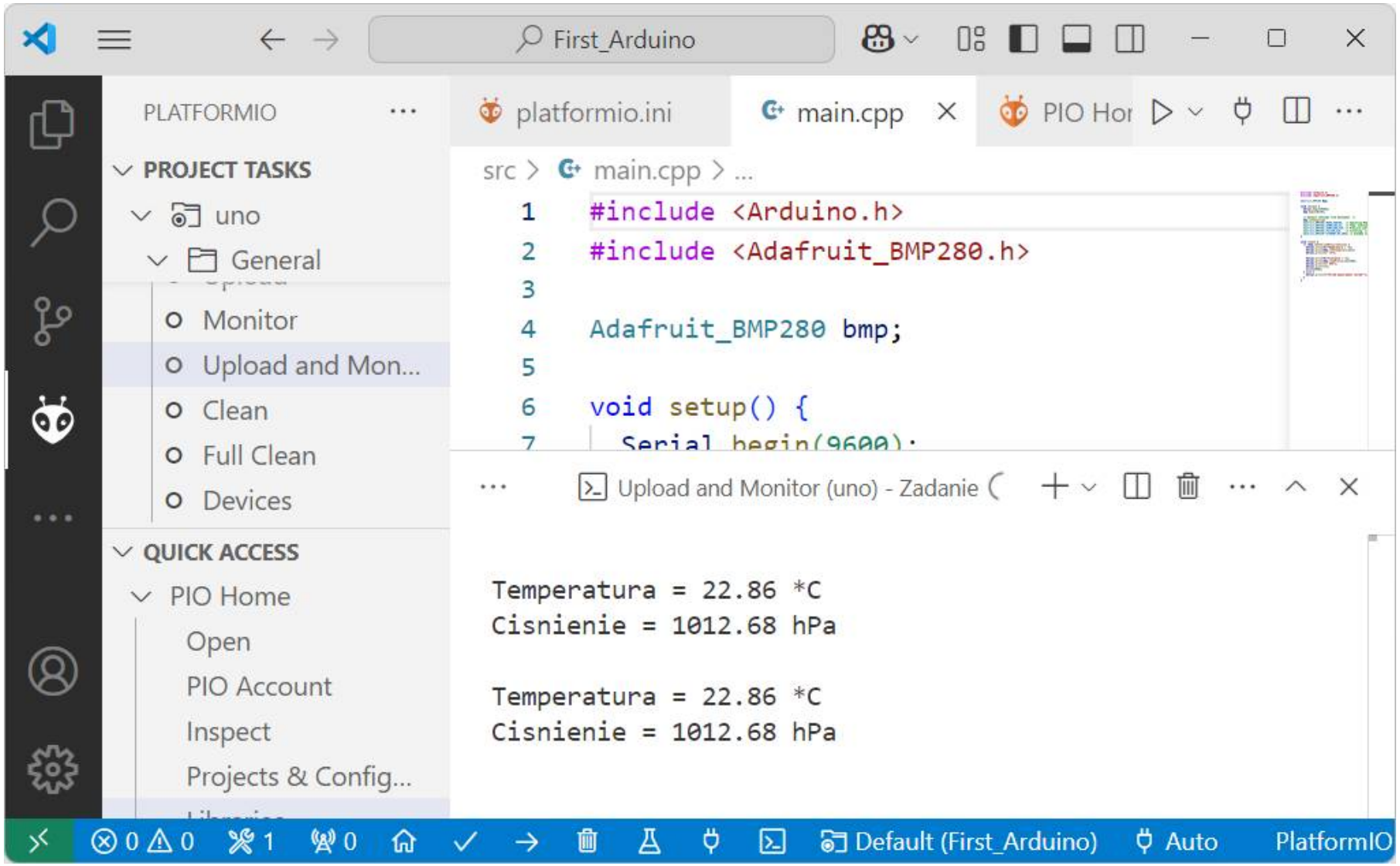
```
#include <Arduino.h>
#include <Adafruit_BMP280.h>

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)

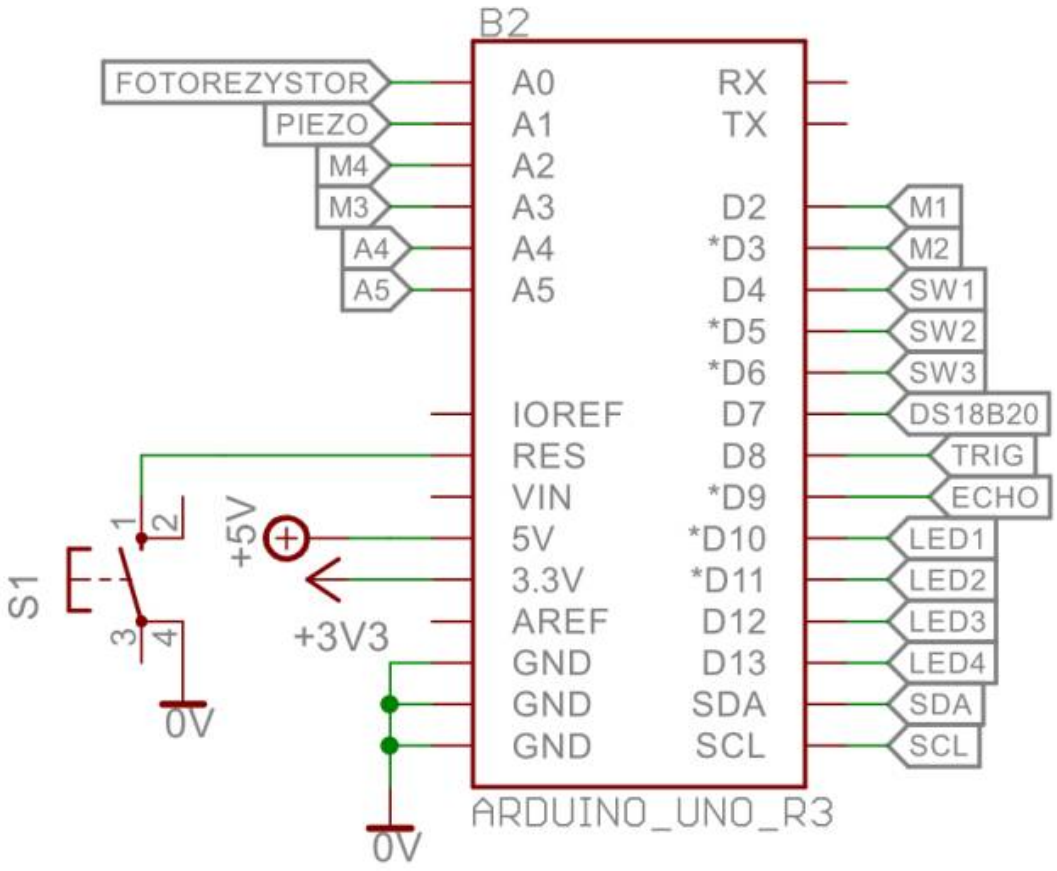
Czujnik ciśnienia BMP280

```
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!");
  }
}
```

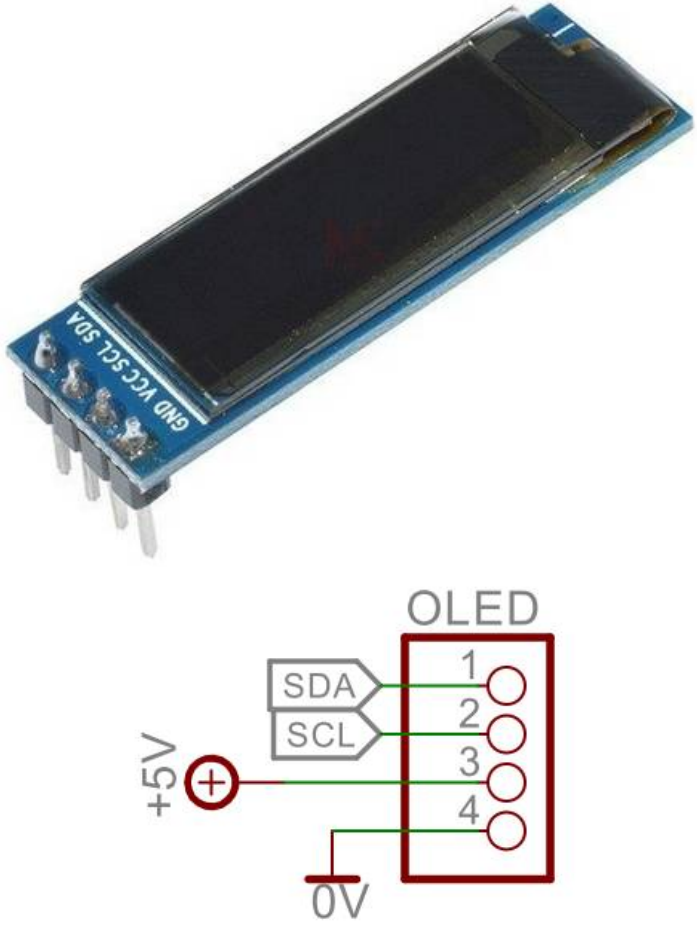
# Arduino - czujnik ciśnienia (BMP280)



# 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 - 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
```

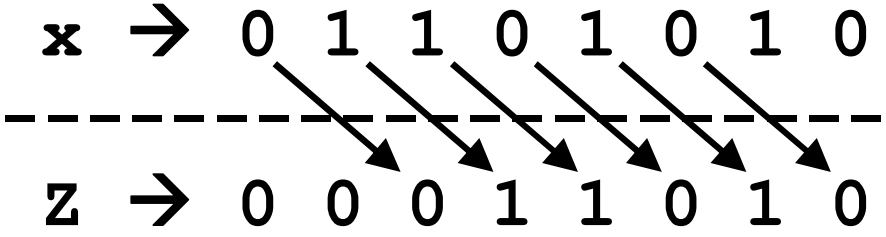
| x | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| y | 0 | 0 | 1 | 1 |
| x ^ y | 0 | 1 | 1 | 0 |

# 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)

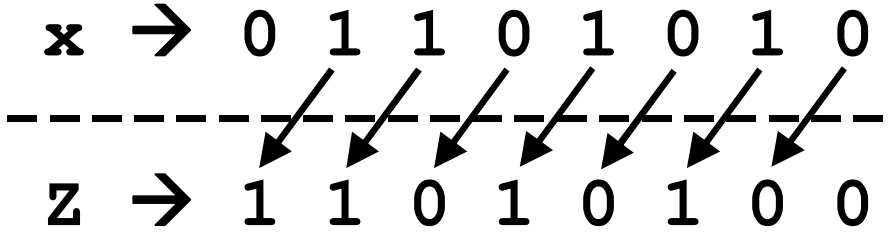


# Język C - operator przesunięcia bitowego (<<)

- 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

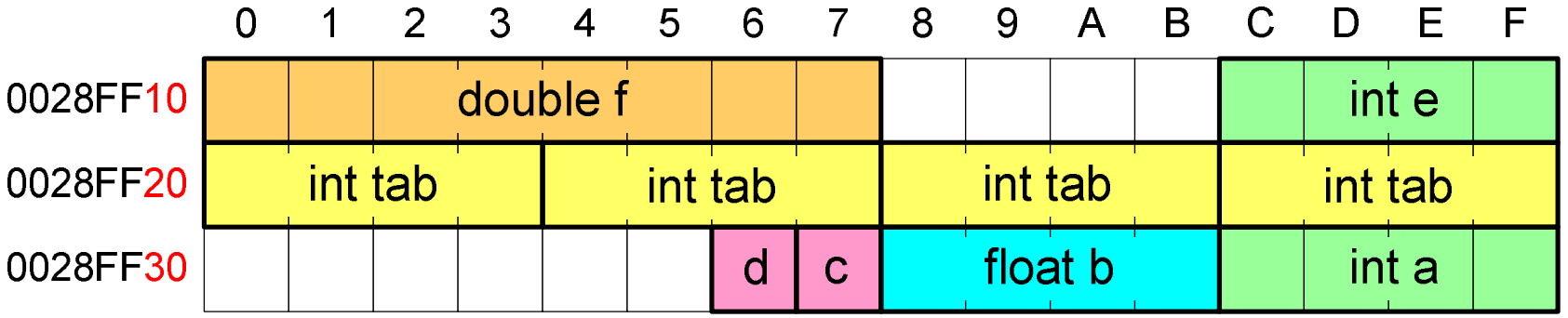


# Co to jest wskaźnik?

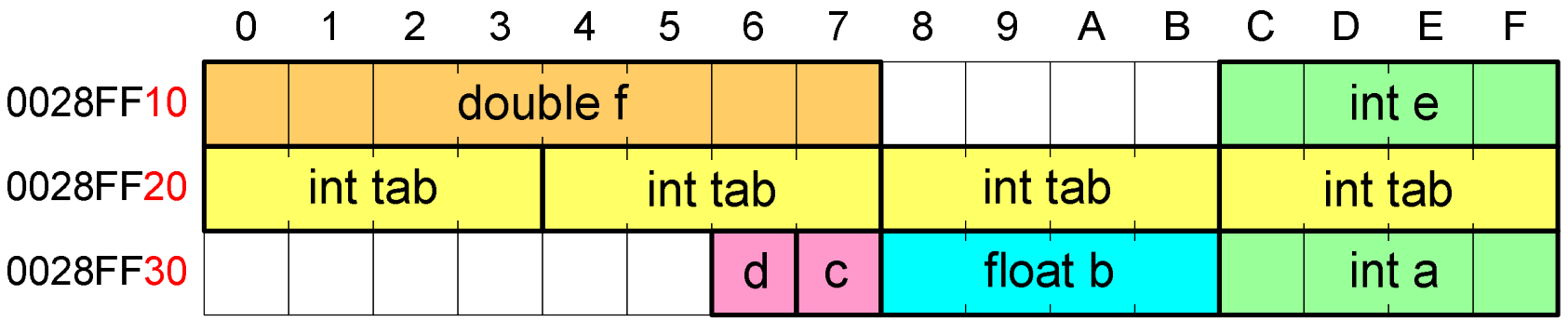
- Wskaźnik - zmienna mogącą zawierać adres obszaru pamięci - najczęściej adres innej zmiennej (obiektu)

```
int a;
float b;
char c, d;
int tab[4], e;
double f;
```

- Zmienne przechowywane są w pamięci komputera



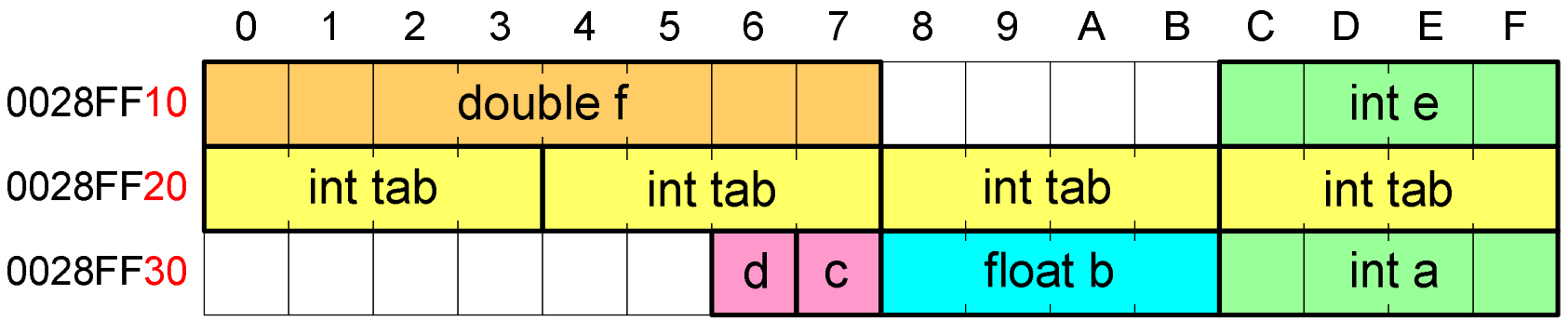
# Co to jest wskaźnik?




- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
printf("Adres zmiennej a:  %p\n", &a);
printf("Adres tablicy tab: %p\n", tab);
```

# Co to jest wskaźnik?




- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
Adres zmiennej a:  0028FF3C
Adres tablicy tab: 0028FF20
```

```
printf("Adres zmiennej a:  %p\n", &a);
printf("Adres tablicy tab: %p\n", tab);
```

# Deklaracja wskaźnika

- Deklarując wskaźnik (zmienną wskazującą) należy podać typ obiektu na jaki on wskazuje
- Deklaracja wskaźnika wygląda tak samo jak każdej innej zmiennej, tylko że jego nazwa poprzedzona jest symbolem gwiazdki (*)

```
typ *nazwa_zmiennej;
```

lub

```
typ* nazwa_zmiennej;
```

lub

```
typ * nazwa_zmiennej;
```

lub

```
typ*nazwa_zmiennej;
```

# Deklaracja wskaźnika

- Deklaracja zmiennej wskaźnikowej do typu int

```
int *ptr;
```

- Mówimy, że zmienna ptr jest typu: wskaźnik do zmiennej typu int
- Do przechowywania adresu zmiennej typu double trzeba zadeklarować zmienną typu: wskaźnik do zmiennej typu double

```
double *ptrd;
```

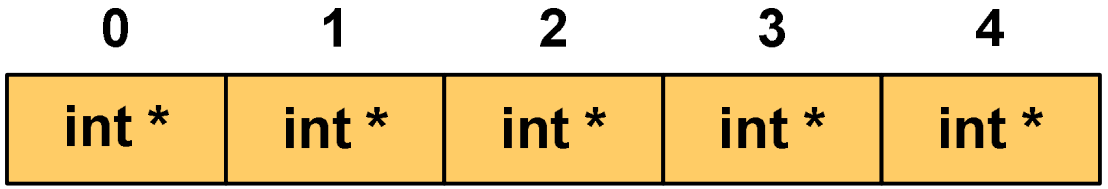
- Można konstruować wskaźniki do danych dowolnego typu łącznie z typami wskaźnik do wskaźnika do...

```
char **wsk;
```

# Deklaracja wskaźnika

- Można deklarować tablice wskaźników - zmienna tab_ptr jest tablicą zawierającą 5 wskaźników do typu int

```
int *tab_ptr[5];
```




- Natomiast zmienna ptr_tab jest wskaźnikiem do 5-elementowej tablicy liczb int

```
int (*ptr_tab)[5];
```

# Deklaracja wskaźnika

- W deklaracji wskaźnika lepiej jest pisać * przy zmiennej, a nie przy typie:

```
int *ptr1;   /* lepiej */
int* ptr2;   /* gorzej */
```

gdyż trudniej jest popełnić błąd przy deklaracji dwóch wskaźników:

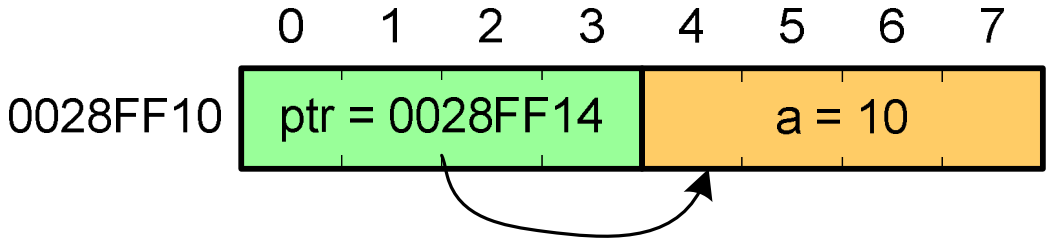
```
int *p1, *p2;
int* p3, p4;
```

- W powyższym przykładzie zmienne p1, p2 i p3 są wskaźnikami do typu int, zaś zmienna p4 jest „zwykłą” zmienną typu int

# Przypisywanie wartości wskaźnikom

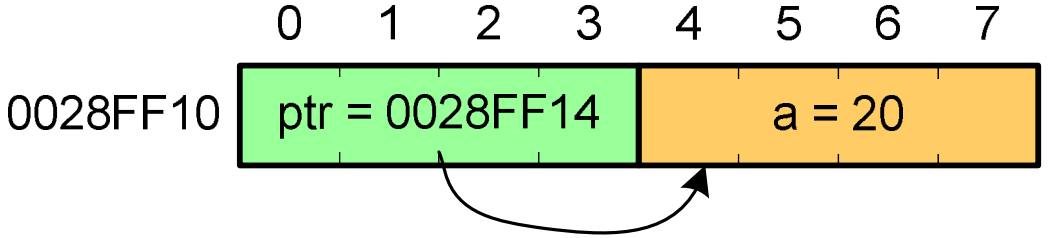
- Wskaźnikom można przypisywać adresy zmiennych
- Adresy takie tworzy się za pomocą operatora pobierania adresu &

```
int a = 10;
int *ptr;
ptr = &a;
```




- Mając adres zmiennej można „dostać się” do jej wartości używając tzw. operatora wyłuskania (odwołania pośredniego) - gwiazdki (*)

```
*ptr = 20;
```



# Wskaźnik pusty

- Wskaźnik pusty to specjalna wartość, odróżnialna od wszystkich innych wartości wskaźnikowych, dla której gwarantuje się nierówność ze wskaźnikiem do dowolnego obiektu
- Do zapisu wskaźnika pustego stosuje się wyrażenie całkowite o wartości zero (0)

```
int *ptr = 0;
```

- Zamiast wartości 0 można stosować makrodefinicję preprocesora NULL, która podczas kompilacji programu zamieniana jest na 0

```
int *ptr = NULL;
```

# Przykład: przypisywanie wartości wskaźnikom

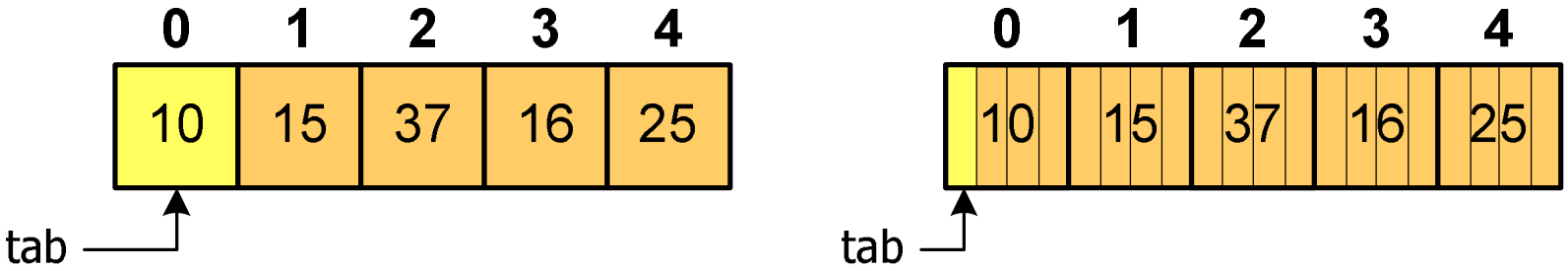
```
#include <stdio.h>

int main(void)
{
   int x = 15;
   int *ptri = NULL;

   printf("x =    %d\n",x);
   printf("ptri = %p\n",ptri);

   ptri = &x;              // przypisanie adresu
   printf("ptri = %p\n",ptri);

   *ptri = *ptri + 10;  // x = x + 10
   printf("x =    %d\n",x);
   printf("x =    %d\n",*ptri);

   return 0;
}
```

```
x =    15
ptri = 0000000000000000
ptri = 00000000010FF960
x =    25
x =    25
```

# Wskaźniki a tablice

- Nazwa tablicy jest jej adresem (dokładniej - adresem elementu o indeksie 0)

```
int tab[5] = {10,15,37,16,25};
```

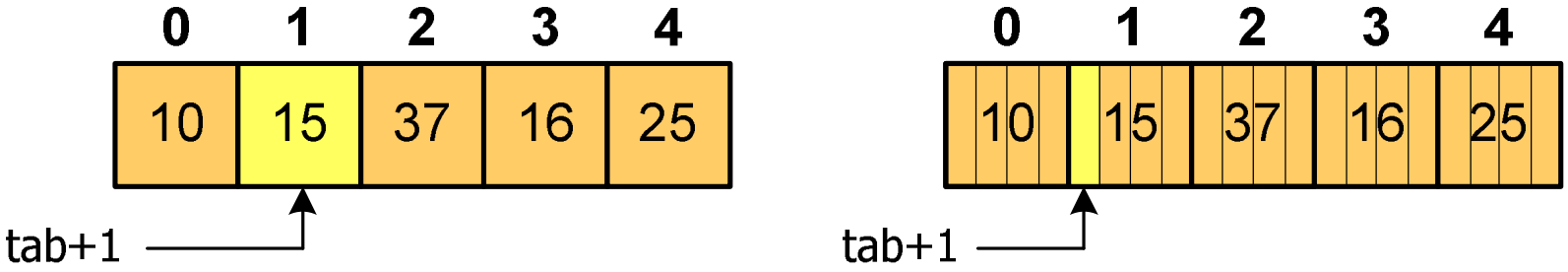



- Zastosowanie operatora * przed nazwą tablicy pozwala „dostać się” do zawartości elementu o indeksie 0

*tab jest równoważne tab[0]

# Wskaźniki a tablice

- Dodanie 1 do adresu tablicy przenosi nas do elementu tablicy o indeksie 1 (przesunięcie o 4 bajty, gdyż int zajmuje 4 bajty)




zatem: *(tab+1) jest równoważne tab[1]

ogólnie: *(tab+i) jest równoważne tab[i]

- W zapisie *(tab+i) nawiasy są konieczne, gdyż operator * ma bardzo wysoki priorytet

x = *tab+1; jest równoważne x = tab[0]+1;

# Wskaźniki a tablice

- Brak nawiasów powoduje błędne odwołania do elementów tablicy

```
int tab[5] = {10,15,37,16,25};
int x;

x = *(tab+2);
printf("x = %d",x);          /*  x = 37 */

x = *tab+2;
printf("x = %d",x);          /*  x = 12 */
```

x = *(tab+2); jest równoważne x = tab[2];

x = *tab+2; jest równoważne x = tab[0]+2;

# Wskaźniki i 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;
```

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

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

# Operacje na wskaźnikach (1)

- Przypisanie - wskaźnikowi można przypisać:
  - adres zmiennej (nazwa zmiennej poprzedzona znakiem &)
  - inny wskaźnik
  - tablicę (nazwa to jej adres)

```
int tab[3] = {1, 2, 3};
int x = 10, *ptr1, *ptr2, *ptr3;

ptr1 = &x;
ptr2 = ptr1;
ptr3 = tab;
```

- Typ adresu i wskaźnika muszą być zgodne

# Operacje na wskaźnikach (2)

- Pobranie wartości (dereferencja)
  - otrzymanie wartości przechowywanej w pamięci, w miejscu wskazywanym przez wskaźnik
  - operator pobrania wartości (dereferencji, wyłuskania): *

```
int x = 10, *ptr, y;

ptr = &x;
y = *ptr;
printf("Wartosc x i y: %d\n",y);
```

```
Wartosc x i y: 10
```

# Operacje na wskaźnikach (3)

- Pobranie adresu wskaźnika
  - tak jak inne zmienne, także wskaźniki posiadają wartość i adres

```
int x = 10, *ptr;

ptr = &x;
printf("Adres zmiennej x:     %p\n",ptr);
printf("Adres wskaznika ptr: %p\n",&ptr);
```

```
Adres zmiennej x:     002CF920
Adres wskaznika ptr: 002CF914
```

# Operacje na wskaźnikach (4)

- Dodanie liczby całkowitej do wskaźnika
  - przed dodaniem liczby całkowitej jest ona mnożona przez liczbę bajtów zajmowanych przez wartość wskazywanego typu

```
int tab[5] = {0,1,2,3,4};

printf("Adres tab:   %p\n",tab);
printf("Adres tab+2: %p\n",(tab+2));
printf("tab[0]:      %d\n",*tab);
printf("tab[2]:      %d\n",*(tab+2));
```

```
Adres tab:   002CFC60
Adres tab+2: 002CFC68
tab[0]:      0
tab[2]:      2
```

# Operacje na wskaźnikach (5)

- Zwiększenie wskaźnika (inkrementacja)
  - do wskaźnika można dodać 1 lub zastosować operator ++
  - wskaźnik będzie pokazywał na kolejny element tablicy

```
int tab[5] = {0,1,2,3,4}, *ptr;

ptr = tab;
printf("tab[0]:  %d\n",*ptr);
ptr++;
printf("tab[1]:  %d\n",*ptr);
ptr = ptr + 1;
printf("tab[2]:  %d\n",*ptr);
```

```
tab[0]:  0
tab[1]:  1
tab[2]:  2
```

# Operacje na wskaźnikach (5)

- Zwiększenie wskaźnika (inkrementacja)
  - do wskaźnika można dodać 1 lub zastosować operator ++
  - wskaźnik będzie pokazywał na kolejny element tablicy

```
int tab[5] = {0,1,2,3,4};

printf("tab[0]:  %d\n",*tab);
tab++;
printf("tab[1]:  %d\n",*tab);
```

error C2105: '++' needs l-value

# Operacje na wskaźnikach (6/7)

- Odjęcie liczby całkowitej od wskaźnika
  - działa analogicznie jak dodanie liczby całkowitej do wskaźnika, ale wskaźnik musi być lewym operandem odejmowania

- Zmniejszenie wskaźnika (dekrementacja)
  - działa analogicznie jak inkrementacja

# Operacje na wskaźnikach (8)

- Odejmowanie wskaźników
  - różnicę między dwoma wskaźnikami oblicza się najczęściej dla wskaźników należących do tej samej tablicy
  - różnica ta określa jak daleko od siebie znajdują się elementy tablicy

```
int tab[5] = {0,1,2,3,4}, *ptr;

ptr = tab + 3;
printf("Roznica:  %d\n",ptr-tab);
```

```
Roznica:  3
```

  - różnica wskaźników należących do dwóch różnych tablic może spowodować błąd w programie

# Operacje na wskaźnikach (9)

- Porównanie wskaźników
  - porównanie może dotyczyć tylko wskaźników tego samego typu
  - w porównaniach stosowane są standardowe operatory:
    <, >, <=, >=, ==, !=

```
int tab[5] = {0,1,2,3,4}, *ptr;

ptr = tab + 2;
ptr--;
--ptr;
if (tab == ptr)
    printf("Ten sam wskaznik\n");
else
    printf("Inny wskaznik\n");
```

```
Ten sam wskaznik
```

# Koniec wykładu nr 7

Dziękuję za uwagę!