



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska

Unia Europejska
Europejski Fundusz Społeczny



Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:

Informatyka
Kod: EDS1B1007

WYKŁAD NR 7

Opracował: dr inż. Jarosław Forenc

Białystok 2019

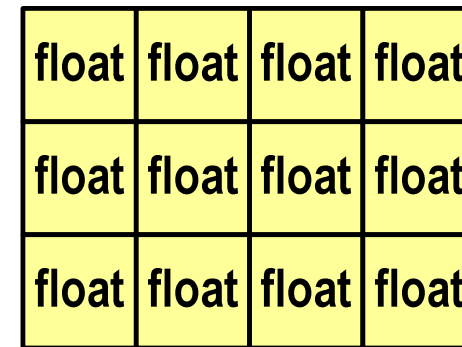
Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Plan wykładu nr 7

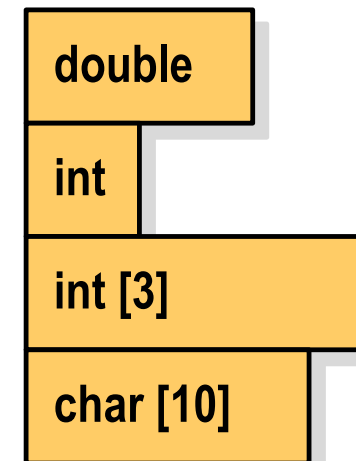
- Struktury, pola bitowe, unie
- Wskaźniki

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;
};
```

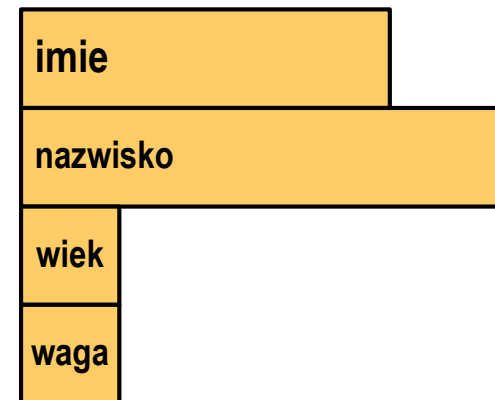
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola struktury)
- Zapisanie danych do struktury wymaga zdefiniowania **zmiennej strukturalnej**

Deklaracja zmiennej strukturalnej

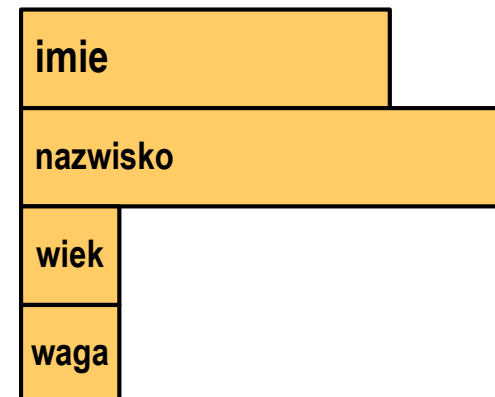
```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal, Nowak;
```

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

Kowal



Nowak



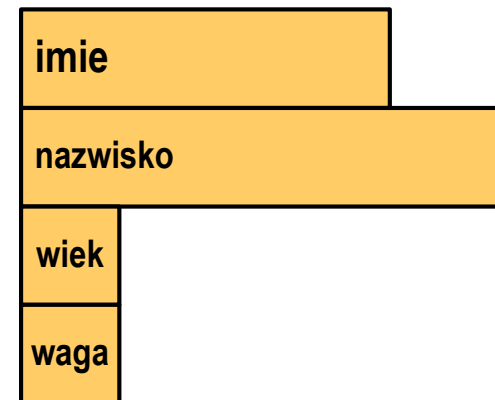
Deklaracja zmiennej strukturalnej

```
#include <stdio.h>

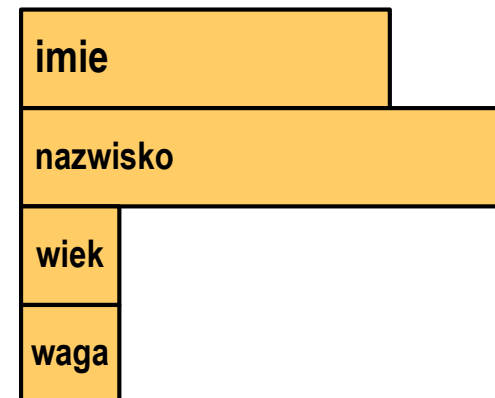
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};

int main(void)
{
    struct osoba Kowal;
    struct osoba Nowak;
    ...
    return 0;
}
```

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 **25** do pola **wiek** zmiennej **Nowak** ma postać

```
Nowak.wiek = 25;
```

- Wyrażenie **Nowak.wiek** traktowane jest jak zmienna typu **int**

```
printf("Wiek: %d\n", Nowak.wiek);  
scanf("%d", &Nowak.wiek);
```


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 **Jan** do pola **imie** zmiennej **Nowak** ma postać

```
strcpy(Nowak.imie, "Jan");
```

- Wyrażenie **Nowak.imie** traktowane jest jak łańcuch znaków

```
printf("Imie: %s\n", Nowak.imie);  
gets(Nowak.imie);
```

Struktury - przykład

```
#include <stdio.h>

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

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

Struktury - przykład

```
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
```

Inicjalizacja zmiennej strukturalnej

- Inicjalizowane mogą być tylko zmienne strukturalne, nie można inicjalizować pól w deklaracji struktury

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

int main(void)
{
    struct osoba Nowak1 = {"Jan", "Nowak", 25, 74};
    ...
}
```

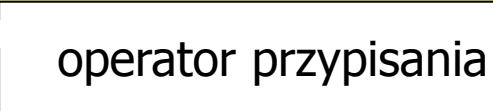
Struktury a operator przypisania (=)

- Struktury tego samego typu można sobie przypisywać (nawet jeśli zawierają tablice)

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

int main(void)
{
    struct osoba Nowak1 = {"Jan", "Nowak", 25, 74};
    struct osoba Nowak2;

    Nowak2 = Nowak1;
}
```



operator przypisania

Struktury w języku 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

```
    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

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

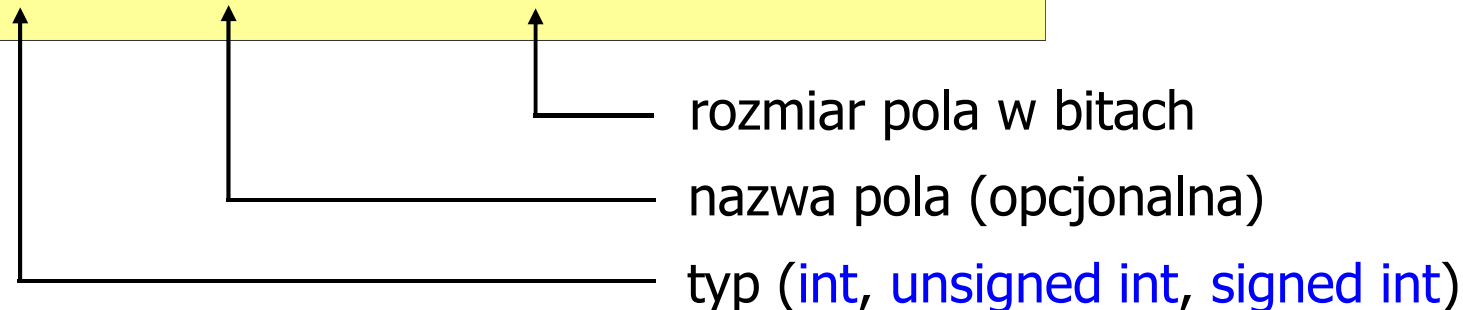
nr		
A	x	y
B	x	y
C	x	y

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


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

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

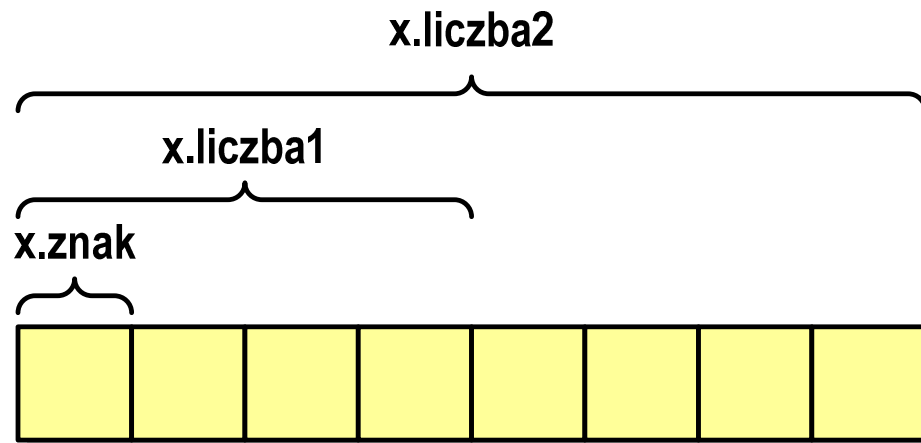
- Do przechowywania wartości w unii należy zadeklarować zmienną

```
union zbior x;
```

Unie

```
union zbior x;
```

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



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

- Rozmiar unii wyznaczany jest przez rozmiar największego jej pola

Unie

```
union zbior x;
```

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

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

- Unię można zainicjować jedynie wartością o typie jej pierwszej składowej

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

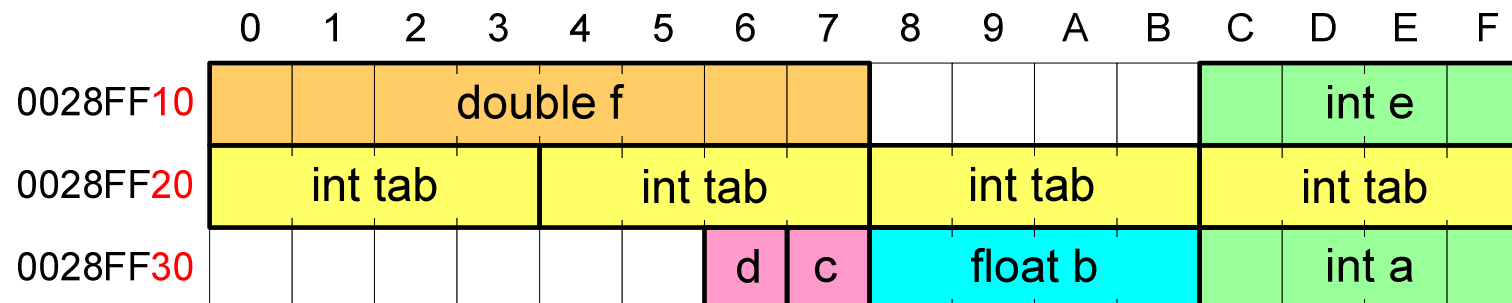
- Unie tego samego typu można sobie przypisywać

Co to jest wskaźnik?

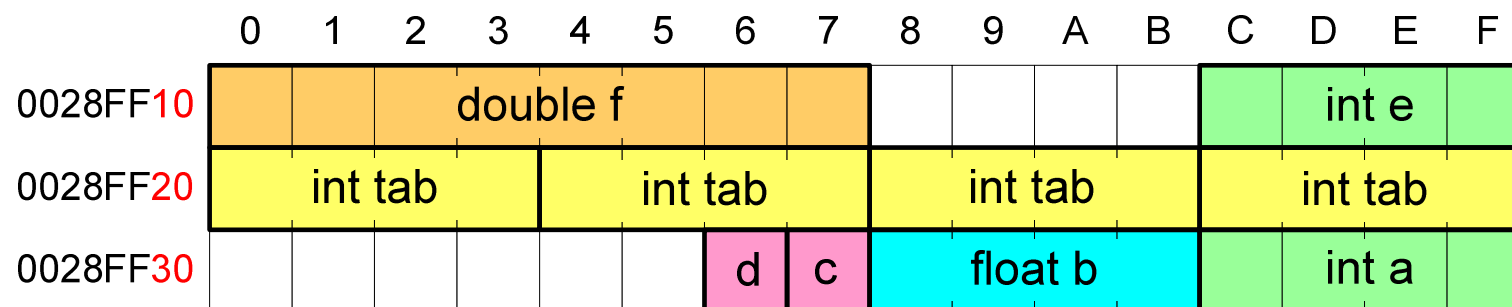
- **Wskaźnik** - zmienna mogąca zawierać adres obszaru pamięci
- najczęściej adres innej zmiennej (obiektu)

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

- Zmienne przechowywane są w pamięci komputera



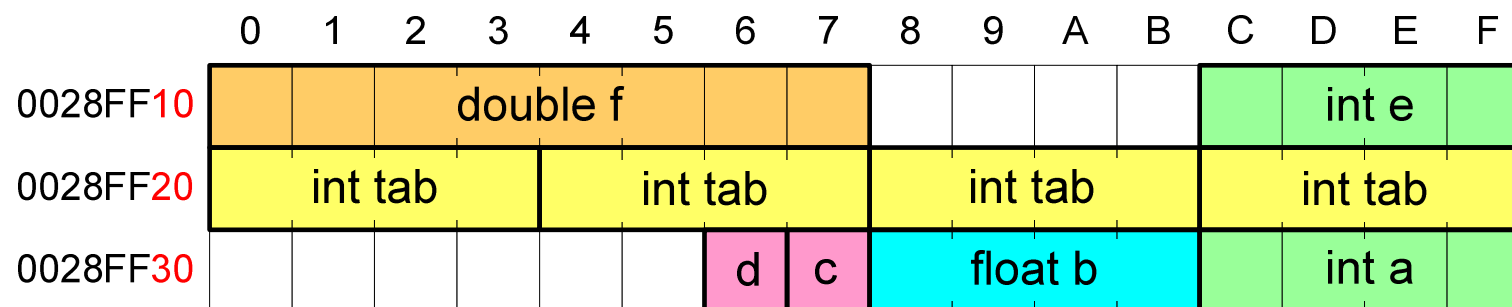
Co to jest wskaźnik?



- 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;
```

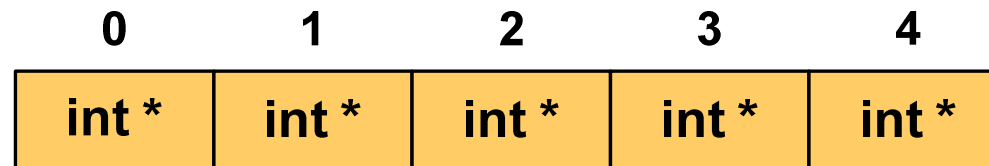
- Można konstruować wskaźniki do danych dowolnego typu łącznie z typami **wskaźnik do wskaźnika do...**

```
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:

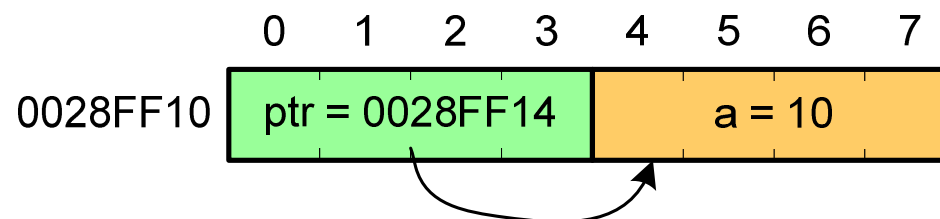
```
int *p1, *p2;  
int* p3, p4;
```

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

Przypisywanie wartości wskaźnikom

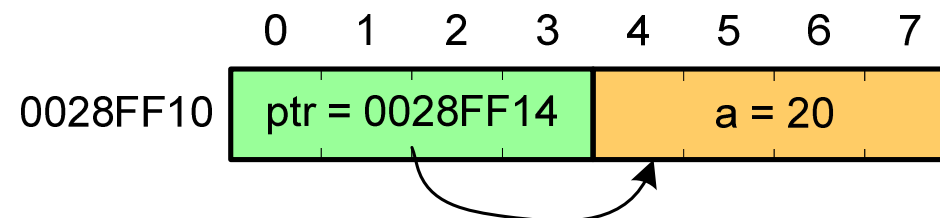
- Wskaźnikom można przypisywać adresy zmiennych
- Adresy takie tworzy się za pomocą operatora pobierania adresu **&**

```
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;
```

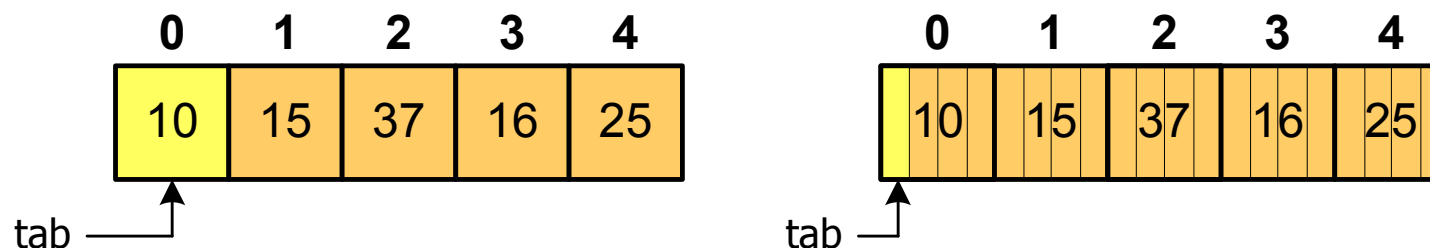
- Zamiast wartości **0** można stosować makrodefinicję preprocesora **NULL**, która podczas kompilacji programu zamieniana jest na **0**

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


Wskaźniki a tablice

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

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

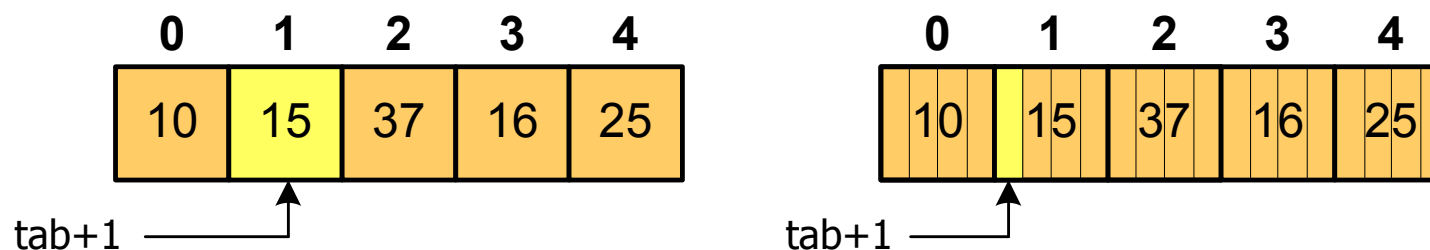


- Zastosowanie operatora `*` przed nazwą tablicy pozwala „dostać się” do zawartości elementu o indeksie 0

`*tab` jest równoważne `tab[0]`

Wskaźniki a tablice

- Dodanie **1** do adresu tablicy przenosi nas do elementu tablicy o indeksie **1**



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

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 $x = tab[0]+2;$

Koniec wykładu nr 7

Dziękuję za uwagę!