

# Programowanie obiektowe (TZ1E2010)

---

Politechnika Białostocka - Wydział Elektryczny

Elektronika i telekomunikacja, semestr II  
studia niestacjonarne I stopnia

Rok akademicki 2020/2021

**Pracownia nr 1 (05.03.2021)**

dr inż. Jarosław Forenc

## Dane podstawowe

- dr inż. Jarosław Forenc
- Politechnika Białostocka, Wydział Elektryczny,  
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki  
ul. Wiejska 45D, 15-351 Białystok  
WE-204
- e-mail: [j.forenc@pb.edu.pl](mailto:j.forenc@pb.edu.pl)
- tel. (0-85) 746-93-97
- <http://jforenc.prv.pl>
  - Dydaktyka - dodatkowe materiały do zajęć
- konsultacje:
  - wtorek, godz. 10:00-11:00, WE-204 / Teams
  - piątek, godz. 12:30-14:30, WE-204 / Teams
  - piątek, godz. 17:00-18:30, WE-204 / Teams (studia zaoczne)
  - niedziela, godz. 08:00-09:00, Teams (studia zaoczne)

## Program przedmiotu (1/3)

1. Zajęcia organizacyjne. Operacje wejścia-wyjścia w języku C++, sterowanie formatem, manipulatory. Struktury, operacje z wykorzystaniem struktur.
2. Funkcje, wywołanie funkcji. Modyfikator const. Przekazywanie argumentów do funkcji (wartość, wskaźnik, referencja) i zwracanie wartości. Przeciążanie funkcji. Szablony funkcji. Wskaźniki i referencje, działania na wskaźnikach. Wskaźniki typu const. Tablice wskaźników.
3. Programowanie strukturalne i obiektowe. Klasa, obiekt, dane i funkcje składowe. Prawa dostępu do składników klasy. Umiejscowienie funkcji składowych klasy.

## Program przedmiotu (2/3)

4. Tworzenie obiektów klasy. Konstruktor. Umieszczenie konstruktora. Użycie wielu konstruktorów. Likwidacja obiektu klasy. Umieszczenie destruktora.
5. Wskaźniki do obiektów klasy. Tworzenie i likwidacja obiektów klasy przy użyciu wskaźników. Wskaźnik this. Przeciążanie funkcji i operatorów.
6. Dziedziczenie. Typy dziedziczenia i dostęp do składowych i funkcji klasy. Konstruktory i destruktory. Dostęp do składowych i funkcji składowych klasy bazowej i pochodnej.

## Program przedmiotu (3/3)

7. Dziedziczenie wielokrotne. Konstruktory i destruktory przy dziedziczeniu wielokrotnym. Eliminacja niejednoznaczności. Dziedziczenie ze wspólnej klasy bazowej.
8. Funkcje wirtualne i klasy abstrakcyjne. Obsługa plików w języku C++.
9. Standardowa biblioteka wzorców STL cz. 1.
10. Standardowa biblioteka wzorców STL cz. 2. Zaliczenie.

## Literatura

1. J. Grębosz: Opus magnum C++11. Programowanie w języku C++. Helion, Gliwice, 2020.
2. J. Grębosz: Symfonia C++ standard. Programowanie w języku C++ orientowane obiektowo t. 1, 2. Wydawnictwo Editions 2000, Kraków, 2015.
3. M. Weisfeld: Myślenie obiektowe w programowaniu. Wydanie V. Helion, Gliwice, 2020.
4. B. Stroustrup: Język C++. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2014.
5. S. Prata: Język C++. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2012.

# Warunki zaliczenia przedmiotu

- Obecność na zajęciach:
  - więcej niż dwie nieusprawiedliwione nieobecności skutkują niezaliczeniem pracowni
- Realizacja w trakcie zajęć zadań przedstawionych przez prowadzącego
- Oddanie i zaliczenie wszystkich sprawozdań z pracowni specjalistycznych:
  - sprawozdanie na ocenę **dostateczną** powinno zawierać cel i zakres ćwiczenia, napisane programy wraz z wynikami ich działania
  - sprawozdanie na ocenę **dobrą** zawiera dodatkowo opis działania programów (np. w postaci komentarzy)
  - sprawozdanie na ocenę **bardzo dobrą** zawiera dodatkowo wnioski, opis badanych mechanizmów i propozycje innych rozwiązań

## Warunki zaliczenia przedmiotu

- Oddanie i zaliczenie wszystkich sprawozdań z pracowni specjalistycznych (cd.):
  - oceny połówkowe są stosowane w przypadku niecałkowitego spełnienia wymagań na ocenę wyższą
  - sprawozdanie powinno zostać oddane na najbliższych zajęciach po zakończeniu tematu, którego dotyczy
  - każdy rozpoczęty tydzień zwłoki w oddaniu sprawozdania skutkuje obniżeniem oceny o 0,5 w stosunku do oceny wyjściowej



## Warunki zaliczenia przedmiotu

- Zaliczenie kolokwium - praktycznego sprawdzianu pisania programu komputerowego:
  - kolokwium odbędzie się na ostatnich zajęciach
  - za kolokwium można otrzymać od 0 do 100 pkt. (min. 51 pkt.)
  - na podstawie otrzymanych punktów wystawiana jest ocena:

Punkty	Ocena	Punkty	Ocena
91 - 100	5,0	61 - 70	3,5
81 - 90	4,5	51 - 60	3,0
71 - 80	4,0	0 - 50	2,0

- Ocena końcowa jest średnią ważoną ocen ze sprawozdań i kolokwium:
  - 60% oceny - ocena z kolokwium
  - 40% oceny - średnia arytmetyczna ocen ze sprawozdań

## Efekty uczenia się

<b>EU1</b>	wyjaśnia strukturę i opracowuje konstrukcję własnych algorytmów z wykorzystaniem instrukcji dostępnych w wybranych językach wysokiego poziomu
<b>EU2</b>	konstruuje własne programy do obliczeń technicznych, oparte na przetwarzaniu obiektowym
<b>EU3</b>	wykorzystuje standardowe biblioteki funkcji, w tym związane z pozyskiwaniem danych z plików
<b>EU4</b>	formułuje i opracowuje własny projekt aplikacji z interfejsem graficznym
<b>EU5</b>	przygotowuje dokumentację dotyczącą opracowanych zadań

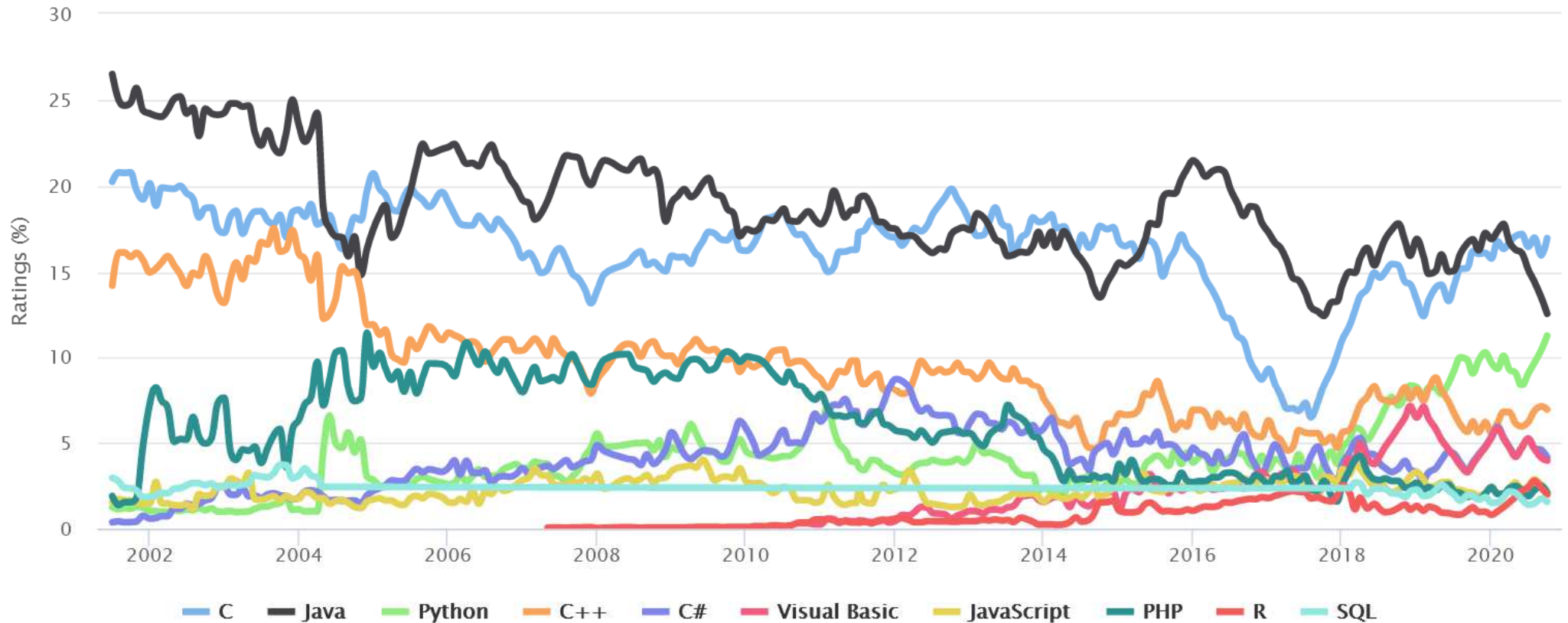
# Język C++

- Stworzony na początku lat 80-tych XX wieku (od 1978 r.) przez **Bjarne Stroustupa** jako obiektowe rozszerzenie języka C
- Pierwsza nazwa: **C z klasami**
- Nazwę **C++** zaproponował Rick Mascitti w 1983 r. (w tym roku po raz pierwszy użyto tego języka poza laboratorium naukowym)
- **++** pochodzi od operatora inkrementacji
- Standard języka C++: rok 1998 - ISO/IEC 14882-1998 „Information Technology - Programming Languages - C++”
- Kolejne wersje standardu: 2003, 2011, 2014, 2017, 2020 (15.12.2020, C++20 - ISO/IEC 14882:2020)

# TIOBE Programming Community Index

TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



# Operacje wejścia-wyjścia w języku C++

- Operacje wejścia-wyjścia nie są zdefiniowane w języku C++
- Operacje te umożliwiają biblioteki standardowo dołączane przez producenta kompilatora:
  - stdio (język C)
  - stream (stara wersja iostream)
  - iostream

## Strumienie:

- Wprowadzanie i wyprowadzanie informacji można potraktować jako strumień bajtów płynących od źródła do ujścia
- Strumienie w C++ realizowane są na zasadzie klas
- Wykorzystanie strumieni wymaga dołączenia pliku nagłówkowego:

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

# Operacje wejścia-wyjścia w języku C++

## ■ Predefiniowane strumienie w C++:

**cout** - związany ze standardowym urządzeniem wyjścia (ekran),  
skrót od ang. **C**-onsole **OUT**-put

**cin** - związany ze standardowym urządzeniem wejścia (klawiatura),  
skrót od ang. **C**-onsole **IN**-put

**cerr** - związany ze standardowym urządzeniem, na które chce się  
wypisywać komunikaty o błędach (ekran) - strumień niebuforowany

**clog** - związany ze standardowym urządzeniem, na które chce się  
wypisywać komunikaty o błędach (ekran) - strumień buforowany

## ■ Za wysyłanie i odbieranie informacji ze strumienia odpowiadają operatory **<<** i **>>**:

**<<** - operator odpowiadający za wysyłanie informacji do strumienia,  
nazywany jest często operatorem **insert** - **wstawienia** (albo **put to**)

**>>** - operator odpowiadający za wczytywanie informacji, nazywany jest  
operatorem **ekstrakcji** (**extract operator**) lub operatorem **get from**

## Wyświetlanie danych

```
int    x = 10, y = 25;
float  z = 1.1234567;
char   txt[10]="Napis";

cout << x;
cout << "x = " << x;
cout << x << y;
cout << x << " " << y;
cout << x << " " << y << endl;
cout << z << endl;
cout << txt << endl;
cout << txt << "\n";
```

```
10
x = 10
1025
10 25
10 25
1.12346
Napis
Napis
```

## Ogólne zasady dotyczące wyświetlania danych

- Liczby całkowite wyświetlane są w systemie dziesiętnym
- Zmienne typów **char**, **unsigned char** wyświetlane są jako pojedyncze znaki
- Liczby zmiennoprzecinkowe typów **float**, **double** wyświetlane są z dokładnością do 6 cyfr (6 cyfr części całkowitej i ułamkowej, bez zbędnych zer)
- **Wskaźniki** wyświetlane są w systemie szesnastkowym
- Zmienne typów **char \***, **unsigned char \*** wyświetlane są jako łańcuchy znaków



# Wczytywanie danych

```
int x, y;  
float z;  
  
cin >> x;  
cin >> x >> y;  
cin >> x >> z;
```

# Ogólne zasady dotyczące wczytywania danych

- Białe znaki (spacja, tabulacja, enter) są ignorowane
- Liczby wczytywane są w systemie dziesiętnym
- Nie można umieszczać spacji pomiędzy znakiem liczby a jej wartością
- Wczytywanie liczby całkowitej jest kończone, gdy napotkany znak nie jest cyfrą
- W liczbach zmiennoprzecinkowych nie może występować spacja w środku
- Wczytywanie tekstów jest kończone po napotkaniu pierwszego białego znaku

## Program w języku C++

```
#include <iostream>

int main()
{
    std::cout << "Witaj swiecie!" << std::endl;
}
```

- **std::** przed nazwami identyfikatorów **cout** i **endl** oznacza, że pochodzą one z biblioteki standardowej (dokładniej - z tzw. **przestrzeni nazw std**)
- **endl** - przejście do nowego wiersza, odpowiada **"\n"** w języku C
- W celu uniknięcia ciągłego pisania **std::** przed nazwami identyfikatorów umieszcza się w programie dyrektywę **using namespace std;**

# Program w języku C++

## Bez dyrektywy using:

```
#include <iostream>

int main()
{
    std::cout << "Witaj swiecie!" << std::endl;
}
```

## Z dyrektywą using:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Witaj swiecie!" << endl;
}
```

## Formatowanie wyjścia

- Metody zmiany sposobu wyświetlania znaków:
  - funkcje `setf`, `unsetf` z klasy `ios` ustawiające odpowiednie flagi
  - funkcje składowe klasy `ios` zmieniające towarzyszące im parametry, np. szerokość, precyzję, itp.
  - manipulatory
  
- Manipulatory (modyfikatory):
  - specjalne wartości, które można wstawić do strumienia po to, aby wywołać zamierzony efekt polegający na zmianie sposobu formatowania
  - manipulatory działają trwale (nie dotyczy to manipulatora `setw`)

# Manipulatory

flush

- opróżnia bufor wyjściowy

endl

- przejście do nowego wiersza - równoważne: `\n` i flush

# Manipulatory

## hex, dec, oct

- określają system liczbowy, w którym są wyświetlane / wczytywane liczby
- **hex** - system szesnastkowy
- **dec** - system dziesiętny
- **oct** - system ósemkowy

```
int x = 100;  
  
cout << x << " " << hex << x << " " << oct << x << endl;  
cout << x << endl;
```

```
100 64 144  
144
```

# Manipulatory

## showbase, noshowbase

- włącza / wyłącza wyświetlanie **0x** na początku liczby w systemie szesnastkowym i **0** na początku liczby w systemie ósemkowym
- działa tylko dla liczb całkowitych

```
int x = 10;  
  
cout << dec << x << " " << showbase << x << noshowbase << endl;  
cout << hex << x << " " << showbase << x << noshowbase << endl;  
cout << oct << x << " " << showbase << x << endl;
```

```
10 10  
a 0xa  
12 012
```



# Manipulatory

## showpos, noshowpos

- włącza / wyłącza pokazywanie znaku liczby dodatniej

```
int x = 10;  
float y = 12.34567;  
  
cout << showpos << x << " " << noshowpos << x << endl;  
cout << showpos << y << " " << noshowpos << y << endl;
```

```
+10 10  
+12.3457 12.3457
```

# Manipulatory

## showpoint, noshowpoint

- włącza / wyłącza pokazywanie nieznaczących zer i kropki dziesiętnej

```
float x = 10;  
cout << showpoint << x << " " << noshowpoint << x << endl;
```

```
10.0000 10
```

# Manipulatory

## fixed, scientific

- **fixed** - włącza notację dziesiętną (tradycyjną)
- **scientific** - włącza notację wykładniczą (naukową)

```
float x = 100.123456;  
  
cout << x << endl;  
cout << fixed << x << endl;  
cout << scientific << x << endl;
```

```
100.123  
100.123459  
1.001235e+002
```

# Manipulatory

## setprecision(int n)

- określa dokładność wyświetlania liczb zmiennoprzecinkowych
- dla „trybu krótkiego” - łączna ilość cyfr przed i po kropce dziesiętnej
- dla **fixed** - ilość miejsc po kropce
- dla **scientific** - dokładność mantysy (ale nie wykładnika)
- działa ciągle
- wymaga dołączenia pliku nagłówkowego **iomanip**

```
float x = 12.123456;  
  
cout << setprecision(5) << x << endl;  
cout << fixed << x << endl;  
cout << scientific << x << endl;
```

```
12.123  
12.12346  
1.21235e+001
```

# Manipulatory

## setw(int n)

- ustawia szerokość wyświetlania liczb lub wczytywania tekstów
- dotyczy tylko najbliższej operacji wejścia-wyjścia
- wymaga dołączenia pliku nagłówkowego **iomanip**

```
int x = 12345;  
  
cout << x << endl;  
cout << setw(10) << x << endl;  
cout << x << endl;
```

12345

12345

12345

# Manipulatory

## setfill(char znak)

- ustawia znak będący wypełnieniem (domyślnie jest to spacja)
- działa ciągle
- wymaga dołączenia pliku nagłówkowego **iomanip**

```
int x = 12345;  
  
cout << setfill('*');  
cout << setw(10) << x << endl;  
cout << setw(10) << x << endl;
```

```
*****12345  
*****12345
```

# Zastosowanie manipulatorów

```
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace std;

#define N 4
#define M 5

int main()
{
    float T[N][M];
    int i, j;

    srand((unsigned int)time(NULL));

    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            T[i][j] = 100*(float)rand()/RAND_MAX - 50;
```

## Zastosowanie manipulatorów

```
cout << fixed << setprecision(3);  
  
for (i=0; i<N; i++)  
{  
    for (j=0; j<M; j++)  
        cout << setw(10) << T[i][j];  
    cout << endl;  
}  
}
```

6.618	-27.541	-35.147	11.235	-28.539
32.482	-45.065	29.177	-42.822	44.375
49.017	46.915	36.261	13.588	12.038
-24.276	6.545	1.527	45.004	6.990



## 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 (**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;
};
```

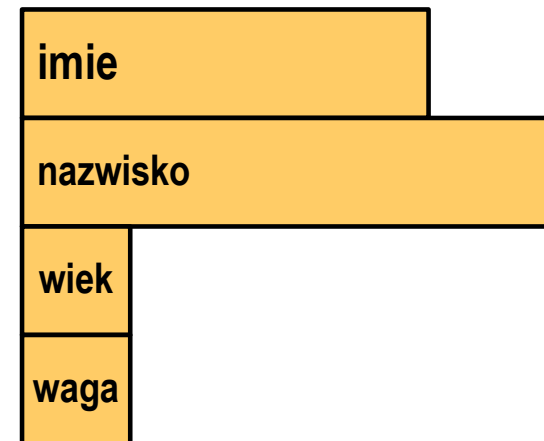
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola struktury)
- Zapisanie danych do struktury wymaga zdefiniowania **zmiennej strukturalnej**

# Deklaracja zmiennej strukturalnej

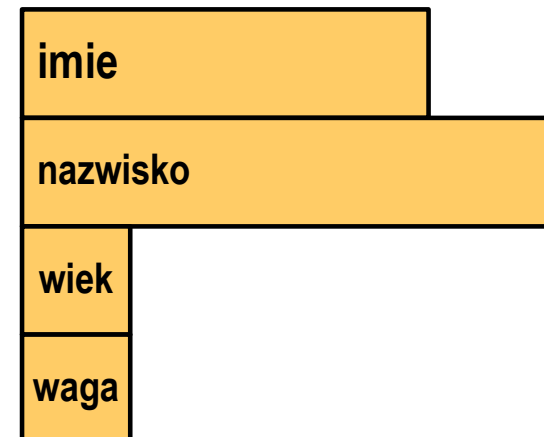
```
struct osoba  
{  
    char imie[15];  
    char nazwisko[20];  
    int wiek, waga;  
} Kowal, Nowak;
```

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

## Kowal



## Nowak



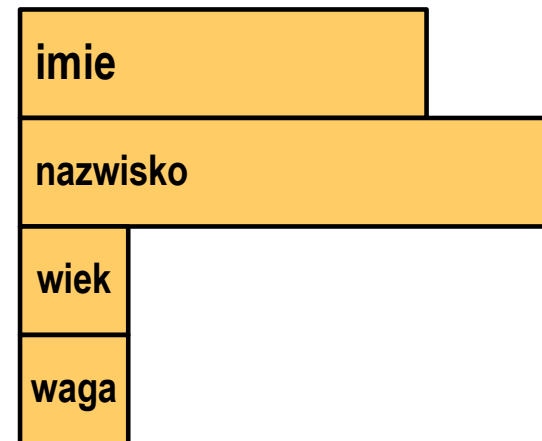
# Deklaracja zmiennej strukturalnej

```
#include <stdio.h>

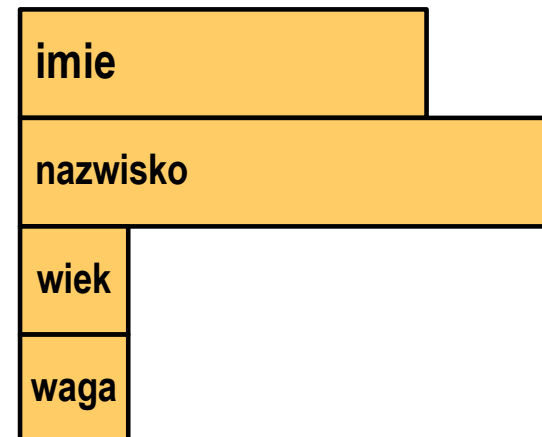
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};

int main(void)
{
    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**

```
cout << "Wiek: " << Nowak.wiek << endl;  
cin >> Nowak.wiek;
```

## 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 **\***

## 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)
{
    osoba Nowak1 = {"Jan", "Nowak", 25, 74};
    ...
}
```