

# Programowanie obiektowe (TZ1E2010)

Politechnika Białostocka - Wydział Elektryczny  
Elektronika i telekomunikacja, semestr II  
studia niestacjonarne I stopnia  
Rok akademicki 2020/2021

Pracownia nr 7 (07.05.2021)

dr inż. Jarosław Forenc

## Przykład: klasa osoba (jeden plik)

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

class osoba
{
private:
    char imie[20];
    char nazwisko[30];
    int wiek;
public:
    void zapisz(char *i, char *n, int w);
    void drukuj();
};

void osoba::zapisz(char *i, char *n, int w)
{
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}
```

## Przykład: klasa osoba (jeden plik)

```
void osoba::drukuj()
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}

int main(void)
{
    osoba os1, os2;

    os1.zapisz("Jan", "Kowalski", 30);
    os2.zapisz("Anna", "Nowak", 25);

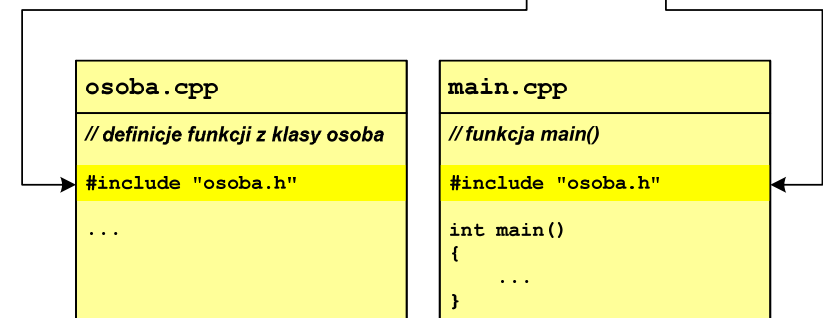
    os1.drukuj();
    os2.drukuj();

    return 0;
}
```

## Klasy a pliki nagłówkowe

- zazwyczaj definicje klas umieszcza się w plikach nagłówkowych \*.h, natomiast definicje funkcji składowych w plikach \*.cpp
- pliki \*.h dołącza się do plików \*.cpp dyrektywą #include "nazwa.h"

```
osoba.h
// deklaracja klasy osoba
class osoba
{
    ...
};
```



## Klasy a pliki nagłówkowe

- w plikach nagłówkowych nie mogą występować definicje funkcji, chyba, że są one umieszczone bezpośrednio w klasie

```
osoba.h

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

class osoba
{
private:
    char imie[20];
    char nazwisko[30];
    int wiek;
public:
    void zapisz(char *i, char *n, int w);
    void drukuj();
};
```

## Klasy a pliki nagłówkowe

- klasę `osoba` wykorzystujemy w pliku `main.cpp`

```
main.cpp

#include "osoba.h"

int main(void)
{
    osoba os1, os2;

    os1.zapisz("Jan", "Kowalski", 30);
    os2.zapisz("Anna", "Nowak", 25);

    os1.drukuj();
    os2.drukuj();

    return 0;
}
```

## Klasy a pliki nagłówkowe

- w celu uniknięcia wielokrotnego dołączania tego samego pliku nagłówkowego stosuje się odpowiednie dyrektywy kompilatora

```
osoba.h

#ifndef _OSOBA_H_
#define _OSOBA_H_

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

class osoba
{
private:
    char imie[20];
    ...
};

#endif
```

## Klasy a pliki nagłówkowe

- w pliku `osoba.cpp` umieszczone są definicje funkcji składowych klasy `osoba`

```
osoba.cpp

#include "osoba.h"

void osoba::zapisz(char *i, char *n, int w)
{
    strcpy(imie, i);
    strcpy(nazwisko, n);
    wiek = w;
}

void osoba::drukuj()
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}
```

## Klasy a pliki nagłówkowe

- w celu uniknięcia wielokrotnego dołączania tego samego pliku nagłówkowego stosuje się odpowiednie dyrektywy kompilatora

```
osoba.h

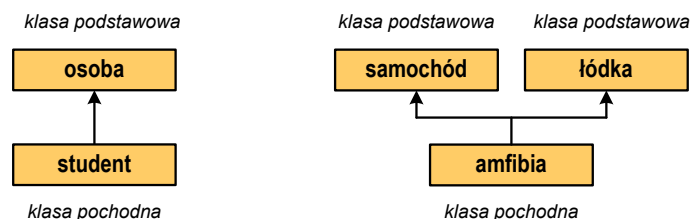
#pragma once

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

class osoba
{
private:
    char imie[20];
    ...
};
```

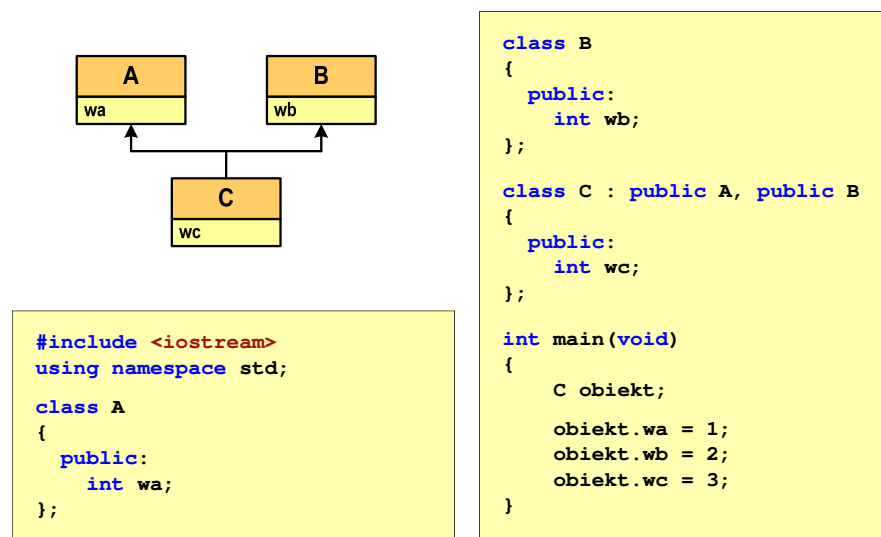
## Dziedziczenie wielokrotne

- **dziedziczenie jednokrotne** - klasa tworzona jest na podstawie jednej klasy podstawowej
- **dziedziczenie wielokrotne (wielobazowe)** - klasa tworzona jest na podstawie więcej niż jednej klasy podstawowej (bazowej)



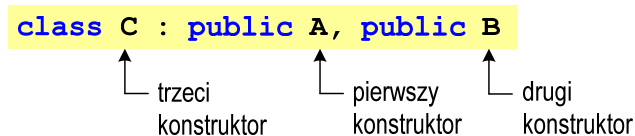
- istnieją języki programowania, w których dziedziczenie wielokrotne nie jest zaimplementowane (np. Java, C#, Object Pascal)

## Przykład: dziedziczenie wielokrotne

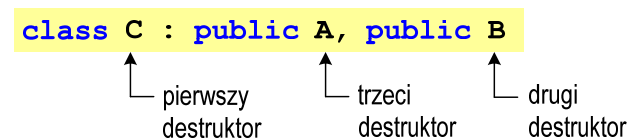


## Dziedziczenie wielokrotne

- kolejność wywołania **konstruktorów** dla obiektu klasy pochodnej wynika z kolejności występowania nazw klas bazowych w deklaracji

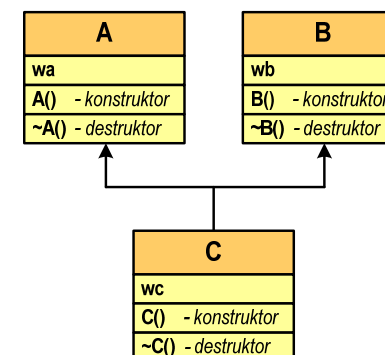


- nie ma znaczenia kolejność umieszczenia konstruktorów klas podstawowych na **liście inicjalizacyjnej** konstruktora klasy pochodnej
- kolejność wywołania **destruktorów** dla obiektu klasy pochodnej jest odwrotna niż konstruktorów



## Przykład: dziedziczenie wielokrotne

- każda klasa (A, B, C) zawiera jedną daną składową, konstruktor, destruktor
- A, B - klasy podstawowe
- C - klasa pochodna



## Przykład: dziedziczenie wielokrotne

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

```
class A
```

```
{
```

```
    int wa;
```

```
public:
```

```
    A(int a = 0) : wa(a)
```

```
    {
        cout << "Konstruktor A()" << endl;
```

```
    }
```

```
    ~A()
```

```
    {
        cout << "Destruktor ~A()" << endl;
```

```
    }
```

```
};
```

konstruktor klasy A

destruktor klasy A

## Przykład: dziedziczenie wielokrotne

```
class B
```

```
{
```

```
    int wb;
```

```
public:
```

```
    B(int b = 0) : wb(b)
```

```
    {
        cout << "Konstruktor B()" << endl;
```

```
    }
```

```
    ~B()
```

```
    {
        cout << "Destruktor ~B()" << endl;
```

```
    }
```

```
};
```

konstruktor klasy B

destruktor klasy B

## Przykład: dziedziczenie wielokrotne

```
class C : public A, public B
```

```
{
```

```
    int wc;
```

```
public:
```

```
    C(int a = 0, int b = 0, int c = 0) : A(a), B(b), wc(c)
```

```
    {
        cout << "Konstruktor C()" << endl;
```

```
    }
```

```
    ~C()
```

```
    {
        cout << "Destruktor ~C()" << endl;
```

```
    }
```

```
};
```

```
int main(void)
```

```
{
```

```
    C obiekt(1,2,3);
```

```
}
```

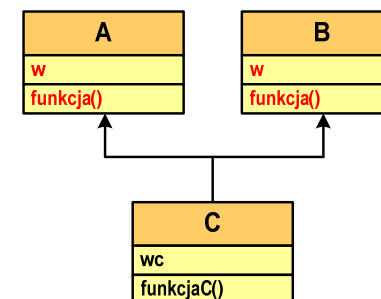
konstruktor klasy C

destruktor klasy c

```
Konstruktor A()
Konstruktor B()
Konstruktor C()
Destruktor ~C()
Destruktor ~B()
Destruktor ~A()
```

## Problemy w dziedziczeniu wielokrotnym

- podstawowy problem w dziedziczeniu wielokrotnym to możliwość występowania **niejednoznaczności**
- w klasach podstawowych występują dane lub funkcje składowe o takich samych nazwach



## Przykład: dziedziczenie wielokrotne (problemy)

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

class A
{
public:
    int w;
    void funkcja(){};
};

class B
{
public:
    int w;
    void funkcja(){};
};
```

```
class C : public A, public B
{
public:
    int wc;
    void funkcjaC(){};
};

int main(void)
{
    C obiekt;

    obiekt.wc = 1;
    obiekt.funkcjaC();
}
```

- w przypadku odwoływania się do danych składowych (`wc`) i funkcji składowych (`funkcjaC`) klasy `C` program skompiluje się i wykona

## Przykład: dziedziczenie wielokrotne (problemy)

- próba odwołania się do danej składowej `w` lub wywołania funkcji `funkcja()` spowoduje błąd kompilacji

```
int main(void)
{
    C obiekt;

    obiekt.w = 1;
    obiekt.funkcja();
}
```

```
main.cpp() : error C2385: ambiguous access of 'w'
           could be the 'w' in base 'A'
           or could be the 'w' in base 'B'
main.cpp() : error C2385: ambiguous access of 'funkcja'
           could be the 'funkcja' in base 'A'
           or could be the 'funkcja' in base 'B'
main.cpp() : error C3861: 'funkcja': identifier not found
```

## Przykład: dziedziczenie wielokrotne (problemy)

- aby odwołania były jednoznaczne należy zastosować operator zasięgu `::`

```
int main(void)
{
    C obiekt;

    obiekt.A::w = 1;
    obiekt.A::funkcja();

    obiekt.B::w = 2;
    obiekt.B::funkcja();
}
```

- w powyższej postaci program skompiluje się i wykona