

# Programowanie obiektowe (TZ1E2010)

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

Pracownia nr 4 (09.04.2021)

dr inż. Jarosław Forenc

## Przykład: klasa osoba z funkcją zapisz()

```
#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 z funkcją zapisz()

```
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;
}
```

```
Jan Kowalski 30
Anna Nowak 25
```

## Konstruktor

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

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

- **konstruktor** służy do nadania wartości początkowych obiektowi i dynamicznego przydzielenia pamięci
- wywołany jest bezpośrednio po utworzeniu obiektu
- nazwa konstruktora jest taka sama jak nazwa klasy
- dla konstruktora nie określamy typu zwracanej wartości (nie może tam wystąpić nawet **void**)
- może mieć kilka wariantów czyli różną liczbę parametrów (**przeładowanie/przeciążenie**)

## Konstruktor

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

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

- „bez” konstruktora:  
`osoba os1;`  
`os1.zapisz("Jan","Kos",30);`
- z konstruktorem:  
`osoba os1("Jan","Kos",30);`  
lub  
`osoba os1=osoba("Jan","Kos",30);`

## Konstruktor domyślny (domniemany)

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

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

osoba::osoba()
{
    strcpy(imie,"");
    strcpy(nazwisko,"");
    wiek = 0;
}
```

- konstruktor, który można wywołać bez żadnego argumentu  
`osoba os1;`  
lub  
`osoba os1 = osoba();`
- jeśli w klasie nie ma takiego konstruktora, to kompilator sam go wygeneruje
- kompilator generuje konstruktor domyślny tylko wtedy, gdy w programie nie ma zdefiniowanego żadnego innego konstruktora

## Konstruktor kopiujący

- **konstruktor kopiujący** służy do skonstruowania obiektu, który jest kopią innego, już istniejącego obiektu tej klasy:

```
klasa::klasa(klasa &);
klasa::klasa(const klasa &);
```

- konstruktor kopiujący nie jest obowiązkowy
- jeśli konstruktor kopiujący nie zostanie zdefiniowany to kompilator wygeneruje go sobie sam (kopiowanie „składnik po składniku”)
- konstruktor kopiujący jest niezbędny, gdy daną składową w klasie jest **wskaźnik**

## Konstruktor kopiujący

- konstruktor kopiujący może być wywołany **jawnie**:

```
klasa obiekt1;
klasa obiekt2 = obiekt1;
```

- konstruktor kopiujący może być wywołany **niejawnie** podczas:
  - przesyłania argumentów do funkcji - jeśli argumentem funkcji jest obiekt klasy, a przesyłanie odbywa się przez wartość
  - zwracania przez funkcję obiektu danej klasy (przez wartość)
- uwaga: w poniższym przykładzie działa **operator przypisania (=)**, a nie konstruktor kopiujący

```
klasa obiekt1, obiekt2;
obiekt2 = obiekt1;
```

## Destruktor

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

osoba::~osoba()
{
    ...
}
```

- ❑ **destruktor** jest wywoływany bezpośrednio przed zniszczeniem obiektu (destruktor można wywołać jawnie - nie spowoduje on jednak usunięcia obiektu)
- ❑ jego nazwa jest taka sama jak nazwa klasy, ale przed jego nazwą umieszczą się znak ~
- ❑ zadaniem destruktora jest „posprzątanie” po obiekcie, np. zwolnienie pamięci
- ❑ dla destruktora nie określamy typu zwracanej wartości

## Przykład: konstruktor i destruktor (1/2)

```
#include <iostream>
#include <cstring>
#pragma warning (disable:4996)

using namespace std;

class test
{
    char name[15];
public:
    test(char *n)
    {
        strcpy(name,n);
        cout << "Konstruktor obiektu: " << name << endl;
    }
    ~test()
    {
        cout << "Destruktor obiektu: " << name << endl;
    }
};
```

## Przykład: konstruktor i destruktor (2/2)

```
int main(void)
{
    test t1("t1"), t2("t2");
    system("pause");
    {
        test t3("t3");
        system("pause");
    }
    system("pause");
    return 0;
}
```

Konstruktor obiektu: t1  
Konstruktor obiektu: t2  
Aby kontynuować, naciśnij . . .  
Konstruktor obiektu: t3  
Aby kontynuować, naciśnij . . .  
Destruktor obiektu: t3  
Aby kontynuować, naciśnij . . .  
Destruktor obiektu: t2  
Destruktor obiektu: t1  
Aby kontynuować, naciśnij . . .

## Lista inicjalizacyjna konstruktora

```
class abc
{
    int a, b, c;
public:
    abc(int aa, int bb, int cc);
};

abc::abc(int aa, int bb, int cc): a(aa), b(bb)
{
    c = cc;
}
```

- ❑ specyfikuje jak należy zainicjować niestyczne składniki klasy
- ❑ **a(aa)** - składnik **a** należy zainicjować wartością wyrażenia w nawiasie (**aa**)
- ❑ lista inicjalizacyjna pojawia się tylko przy definicji konstruktora
- ❑ kolejność umieszczania elementów na liście inicjalizacyjnej nie ma znaczenia

## Dynamiczny przydział pamięci na obiekt

```
class osoba
{
public:
    char imie[20];
    char nazwisko[30];
    int wiek;
};

int main(void)
{
    osoba os, *ptr_os;
    ptr_os = new osoba;
    os.wiek = 25;
    ptr_os->wiek = 25;
    delete ptr_os;
}
```

- operator **new** można zastosować do dynamicznego tworzenia obiektu w trakcie wykonywania programu
- pamięć utworzonego obiektu należy zwolnić za pomocą operatora **delete**

## Przykład: dynamiczny przydział pamięci (1/3)

```
#include <iostream>
#include <cstring>

#pragma warning (disable:4996)

using namespace std;

class osoba
{
private:
    char *imie;
    char *nazwisko;
    int wiek;
public:
    osoba(char *i, char *n, int w);
    ~osoba();
    void drukuj();
};
```

wskaźniki

## Przykład: dynamiczny przydział pamięci (2/3)

```
osoba::osoba(char *i, char *n, int w)
{
    imie = new char[strlen(i)+1];
    nazwisko = new char[strlen(n)+1];
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}

osoba::~osoba()
{
    delete [] imie;
    delete [] nazwisko;
}

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

konstruktor

destruktor

funkcja drukuj()

## Przykład: dynamiczny przydział pamięci (3/3)

```
int main(void)
{
    osoba os1("Jan", "Kos", 30);
    os1.drukuj();
}
```

Jan Kos 30

- w kolejnym przykładzie zdefiniujemy funkcję **drukuj()** jako funkcję zaprzyjaźnioną z klasą
- w definicji klasy: **friend void drukuj(osoba ktos);**
- definicja funkcji: **void drukuj(osoba ktos)**

```
{
    cout << ktos.imie << " " << ktos.nazwisko;
    cout << " " << ktos.wiek << endl;
}
```
- wywołanie funkcji: **drukuj(os1);**

## Przykład: konstruktor kopiujący (1/3)

```
#include <iostream>
#include <cstring>

#pragma warning (disable:4996)

using namespace std;

class osoba
{
private:
    char *imie;
    char *nazwisko;
    int wiek;
public:
    osoba(char *i, char *n, int w);
    osoba(const osoba &os);
    ~osoba();
    friend void drukuj(osoba ktos);
};
```

konstruktor kopiujący

## Przykład: konstruktor kopiujący (2/3)

```
osoba::osoba(char *i, char *n, int w)
{
    imie = new char[strlen(i)+1];
    nazwisko = new char[strlen(n)+1];
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}

osoba::osoba(const osoba &os)
{
    imie = new char[strlen(os.imie)+1];
    nazwisko = new char[strlen(os.nazwisko)+1];
    strcpy(nazwisko,os.nazwisko);
    wiek = os.wiek;
}
```

konstruktor kopiujący

## Przykład: konstruktor kopiujący (3/3)

```
osoba::~osoba()
{
    delete [] imie;
    delete [] nazwisko;
}

void drukuj(osoba ktos)
{
    cout << ktos.imie << " " << ktos.nazwisko;
    cout << " " << ktos.wiek << endl;
}

int main(void)
{
    osoba os1("Jan", "Kos", 30);
    drukuj(os1);
}
```

Jan Kos 30