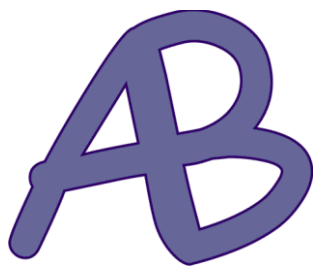


*dr Artur Bartoszewski*  
*Katedra Informatyki*  
*UTH Radom*



## **Wykład 4:** **Klasy cz. 3**



# Przesłanianie i przeciążanie metod w klasach

## Przeciążanie metod

**Przeciążanie metod** (ang. Overloading) - pozwala na tworzenie metod o takich samych nazwach, ale różniących się listą parametrów, tzn. liczbą lub typem parametrów z którymi metoda jest wywoływana.

```
class Klasa
{
    public:
        void funkcja();
        void funkcja(int a);
        void funkcja(float a);
        void funkcja(int a, int b);
};
```

### Nieprawidłowe jest:

- utworzenie w jednej klasie dwóch metod o identycznej nazwie i przyjmującej takie same parametry,
- metod o takiej samej nazwie i parametrach, ale różniące się **tylko** zwracanym typem.

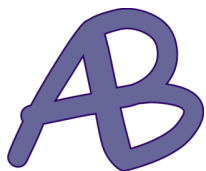
# Przeciążanie metod



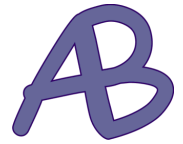
```
class dodawanie{  
    int dodaj(int a, int b){  
        return a+b;  
    }  
  
    double dodaj(double a, double b){  
        return a+b;  
    }  
}
```



```
class dodawanie{  
    int dodaj(int a, int b){  
        return a+b;  
    }  
  
    double dodaj(int a, int b){  
        return a+b;  
    }  
}
```



## **Konstruktor i destruktor** (część II)



## Konstruktor i destruktor

---

### Jawne wywołanie konstruktora

Obiekt może być też stworzony przez jawne wywołanie konstruktora.

W efekcie otrzymujemy obiekt, który nie ma nazwy, a czas jego życia ogranicza się do wyrażenia, w którym go użyto.

`nazwa_klasy(argumenty)`

**Uwaga:** nie stosujemy zapisu:

`obiekt.funkeja_składowa(argumenty)`

Konstruktor nie jest wywoływany na rzecz jakiegoś obiektu, bo ten obiekt jeszcze nie istnieje. Zadaniem konstruktora jest go utworzyć.

## Konstruktor i destruktory

---

### Jawne wywołanie destruktora

Należy podać całą jego nazwę. Jawne wywołanie destruktora nie może się zacząć od ~(wężyka) i wcześniej musi być albo obiekt, na rzecz którego jest wywoływany i kropka lub wskaźnik do obiektu „->”

`obiekt.~klasa();`

`wskaznik->~klasa();`



## Konstruktor i destruktor

---

### Konstruktor domyślny

Konstruktor domyślny to konstruktor, który można wywołać bez żadnego argumentu.

```
class klasa {  
    public :  
        klasa() ;    //konst.    domyślny  
};
```





# Konstruktor i destruktor

---

## Konstruktor domyślny

Konstruktor domyślny może posiadać dowolną ilość parametrów, jednak wszystkie muszą mieć zdefiniowaną wartość domyślną (żeby nie trzeba było podawać ich wartości podczas inicjacji klasy).

```
class Klasa
{
public:
    Klasa(int x = 10, char c = 'q', string s = "tekst")
    {
    }
};
```

# Konstruktor i destruktor

## Konstruktor domyślny z parametrami domyślnymi

```
class Osoba {  
public:  
    string imie;  
    int wiek;  
  
    Osoba(string imie = "NN", int wiek = 0) {  
        this->imie = imie;  
        this->wiek = wiek;  
    }  
};  
  
int main()  
{  
    Osoba *ktos = new Osoba();  
    Osoba ktos2("Karol", 22);  
    cout << ktos->imie << endl; //wyswietli NN  
    cout << ktos2.imie << endl;  //wyswietli Kar  
    return 0;  
}
```

# Konstruktor i destruktor

## Przeciążanie konstruktorów

Jedna klasa może posiadać kilka konstruktorów różniących się listą parametrów (ich liczbą i/lub typem).

```
class klasa {  
    public :  
        klasa(void) ; //konst. domniemany  
        klasa(int) ;  
        klasa(float)  
        klasa(int, int) ;  
};
```

# Konstruktor i destruktor

## Przeciążanie konstruktorów

W klasie może istnieć tylko jeden konstruktor domyślny, to znaczy taki który można wywołać bez parametrów

```
class klasa {  
    public :  
        // klasa(void) ; //konst. domniemany  
        klasa(int);  
        klasa(float)  
        klasa(int, int);  
        klasa (int a=5, char *s = NULL);  
        // to też może być konst. domniemany ale tylko jeden w klasie  
};
```



## Konstruktor i destruktor

---

### Lista inicjalizacyjna konstruktora

Czasami zachodzi potrzeba zainicjowania zmiennej w trakcie tworzenia klasy, a nie po jej utworzeniu. Korzystamy wtedy z tak zwanej **listy inicjalizacyjnej konstruktora**.

**Lista inicjalizacyjna** to lista oddzielonych przecinkami identyfikatorów pól (składowych) z podanymi w nawiasach okrągłych argumentami dla konstruktorów obiektów będących składowymi tworzonego obiektu. Zwykle są to jednocześnie argumenty formalne definiowanego konstruktora, choć nie musi tak być.

# Konstruktor i destruktork

## Lista inicjalizacyjna konstruktora

```
class Osoba {  
public:  
    int wiek;
```

```
    Osoba(int WIEK) {  
        wiek = WIEK;  
    }
```

```
};
```

konstruktor  
wieloargumentowy

```
class Osoba {  
public:  
    int wiek;
```

```
    Osoba(int WIEK) : wiek(WIEK) {  
    }
```

```
};
```

konstruktor  
wieloargumentowy

lista  
inicjalizacyjna

p-programowanie.pl

Wartość zmiennej WIEK, która jest **argumentem konstruktora** przypisywana jest zmiennej wiek, która jest składnikiem klasy.

Po przecinku można wypisać kolejne **inicjalizowane** składowe.

# Konstruktor i destruktor

---

Taki zapis ma kilka bardzo istotnych zalet.

1. **Jest szybszy** - różnice są znaczne gdy przyjdzie do wykonywania pomiarów czasowych.
2. **Jest czytelniejszy** - programista nie musi analizować zawartości konstruktora, by wiedzieć jaką domyślną wartością zostanie zainicjowana klasa.
3. Umożliwia inicjowanie zmiennych zdefiniowanych jako **stałe** (const).
4. Umożliwia inicjowanie zmiennych zdefiniowanych jako **referencje** (tylko tą metodą można zainicjować zmienną zadeklarowaną np. tak: `int & zmienna;`).
5. **Jest metodą stosowaną przy dziedziczeniu klas.**

## Konstruktor i destruktor

---

### Lista inicjalizacyjna konstruktora

- ✓ Zwróćmy uwagę, że w przypadku użycia listy inicjalizacyjnej ciało konstruktora jest puste. Można oczywiście umieścić w nim jakiejś instrukcje.
- ✓ Niektóre składowe mogą być inicjalizowane poprzez konstruktor a inne poprzez listę inicjalizacyjną.
- ✓ Jeśli w klasie tylko deklarujemy konstruktor, a jego definicję podajemy poza klasą, to w deklaracji listy inicjalizacyjnej nie umieszczamy.

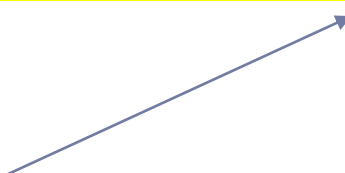


# Konstruktor i destruktor

---

```
class Klasa
{
private:
    int x;
    char znak;
    string napis;
public:
    Klasa(int X, char ZNAK, string NAPIS) : x(X), znak(ZNAK), napis(NAPIS){}
};
```

**Lista inicjalizująca** przekazuje parametry konstruktora (w nawiasach - pisane wielkimi literami) do pól klasy (pisane małymi literami)



Ciało konstruktora może być puste lecz musi istnieć

# Konstruktor i destruktor

```
class Klasa
{
private:
    int x;
    char znak;
    string napis;
```

```
public:
```

```
    Klasa(int X, char ZNAK, string NAPIS); //nagłówek konstruktora
};
```

```
Klasa::Klasa(int X, char ZNAK, string NAPIS) : x(X), znak(ZNAK), napis(NAPIS)
{
}
```

Treść konstruktora może być opisana poza klasą.

Wewnątrz klasy umieszczamy tylko nagłówek konstruktora, bez listy inicjalizującej

Charakterystyczna konstrukcja konstruktora opisanego poza klasą.

Powtórzenie nazwy klasy wynika stąd, że konstruktor nazywa się tak samo jak klasa.

# Konstruktor i destruktor

```
class test
```

```
{  
private:  
    int x;  
    char znak;  
    string napis;
```

```
public:
```

```
test() : x(100), znak('q'), napis("napis") {}
```

```
test(int X, char ZNAK, string NAPIS) : x(X), znak(ZNAK), napis(NAPIS){}  
};
```

Listy inicjalizującej mającej można użyć także do stworzenia konstruktora domyślnego, wywoływanego bez parametrów.

W nawiasach listy umieszczamy wartości które mają być wstawione do odpowiednich pól klasy

## Literatura:

---

W prezentacji wykorzystano przykłady i fragmenty:

- Grębosz J. : ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zajrzeć także do:

- Sokół R. : ***Microsoft Visual Studio 2012 Programowanie w Ci C++***, Helion.
- Kernighan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Grębosz J. : ***Pasja C++***, Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne