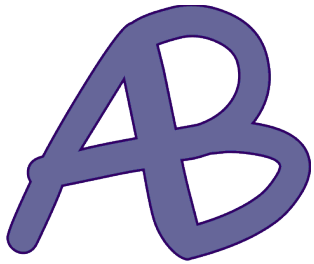
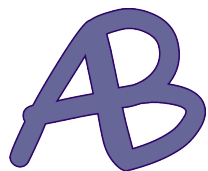


Wykład 3: Klasy cz. 2



- ✓ Konstruktor i destruktork
- ✓ Przesłanianie składników klas
- ✓ Wskaźnik *this*



Konstruktor i destruktor (część 1)

Temat konstruktora będzie jeszcze poruszany,
szczególnie w kontekście dziedziczenia i
polimorfizmu

Konstruktor i destruktork

Dla przykładu przeanalizujemy zainicjalizowanie pola obiektu jakąś wartością.

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

class numer {
    int  liczba ;
public:
    void schowaj(int x) { liczba = x ; }
    int zwracaj() { return liczba ; }
} ;

int main()
{
    numer skrytka;
    skrytka.schowaj(10);
    cout << "w skrytce jest liczba " << skrytka.zwracaj() << endl;
    return 0;
}
```

Potrzebne są aż dwie operacje

Tworząc obiekt **skrytka** (instancję klasy numer) wykonujemy dwie operacje:

1. Tworzymy obiekt klasy **numer** (w polu **liczba** zawiera on losową wartość)
2. Przy pomocy metody **schowaj()** nadajemy wartość polu **liczba**)

Obie te operacje wykonać możemy za jednym razem - tworząc obiekt można od razu wypełnić danymi jego pola. Umożliwia to **konstruktor**.

Konstruktor i destruktork

Tym razem dodamy do obiektu **konstruktor**

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

class numer {
    int  liczba ;
public:
    numer (int x) {liczba = x;}
    void schowaj(int x)  { liczba = x ; }
    int zwracaj()  { return liczba ; }
};

int main()
{
    numer skrytka(10);
    cout << "w skrytce jest liczba " << skrytka.zwracaj() << endl;
    return 0;
}
```

konstruktor - metoda o takiej samej nazwie jak klasa, uruchamiana automatycznie po utworzeniu obiektu

Objaśnienie składni nowych elementów na kolejnych stronach

Konstruktor i destruktor

- ✓ Konstruktor to metoda klasy, która jest wywoływana podczas tworzenia jej instancji.
- ✓ Konstruktor nazywa się tak samo jak klasa.
- ✓ Przed konstruktorem nie ma żadnego określenia typu wartości zwracanej. Nie może być tam nawet typu **void**. Po prostu nie stoi tam nic.
- ✓ W konstruktorze nie może wystąpić instrukcja **return**.

Konstruktor to metoda, która jest uruchamiana przy tworzeniu każdego obiektu klasy. Dzięki konstruktorowi jesteśmy w stanie zainicjalizować pola w klasie. Konstruktor może też wykonać obliczenia lub operacje, które powinny być wykonane automatycznie przy tworzeniu każdego nowego obiektu.

W czasie wykonywania konstruktora obiekt już istnieje, to znaczy został utworzony w pamięci, co za tym idzie zostały już utworzone wszystkie pola klasy - konstruktor ma więc do nich dostęp (może je modyfikować).

Konstruktor i destruktor

Konstruktor bez parametrów

```
class klasa
{
    int pole;
public:
    klasa() //konstruktor
    {
        pole = 10;
    }
};
```

Jeżeli klasa posiada konstruktor **bez parametrów** – nazywany też **konstruktorem domyślnym**, zostanie on wywołany w chwili tworzenia obiektu klasy, nawet bez naszej ingerencji.

```
klasa obiekt01;
```

Konstruktor i destruktor

Konstruktor z paramatrami


```
class klasa
{
    public:
        int pole;
    public:
        klasa(int wartosc)
        {
            //konstruktor
            pole = wartosc;
        }
};

int main()
{
    klasa obiekt01(101);
    return 0;
}
```

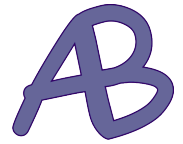
Jeżeli klasa posiada konstruktor z **parametrami** należy wywołać go jawnie i podać wartości oczekiwanych parametrów.

```
klasa obiekt01(101);
```

~~klasa obiekt01;~~



Takie wywołanie jest w tej sytuacji błędne.



Konstruktor i destruktor

DESTRUKTOR

Przeciwieństwem konstruktora jest **destruktor** - funkcja składowa wywoływana wtedy, gdy obiekt danej klasy ma być zlikwidowany.

Destruktor nazywa się tak samo, jak klasa z tym, że przed nazwą ma znak **~** (**tylda**). Podobnie jak konstruktor - nie ma on określenia typu zwracanego.

Konstruktor i destruktor

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

class numer {
    int    liczba ;
public:
    numer (int x) {liczba = x;}
    void schowaj(int x)  { liczba = x ; }
    int zwracaj()  { return liczba ; }
    ~numer () {cout << "no to pa!";}
};

int main()
{
    numer skrytka(10);
    cout << "w skrytce jest liczba " << skrytka.zwracaj() << endl;
    return 0;
}
```

Destruktor wywołany
zostanie przed
zamknięciem
programu, pomimo, że
nie istnieje jego jawne
wywołanie

Konstruktor i destruktor

```
class Liczba
{
private:
    int x;
public:
    Liczba(int px) { x = px; } //brak wartosci domyslnej
    void setX(int px) { x = px; }
    int getX() { return x; }
};
```

Utworzenie tablicy obiektów, które **nie posiadają konstruktora domyślnego** jest niemożliwe

```
Liczba tab[100]
```

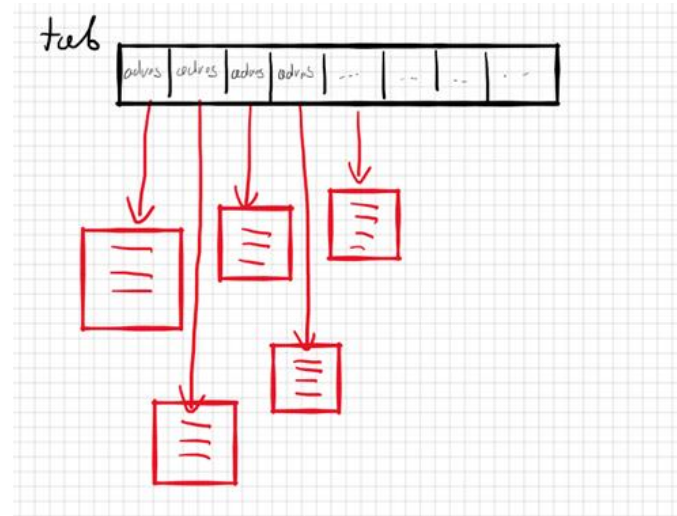
klasa "Liczba" nie ma konstruktora domyślnego C/C++(291)

[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

```
Liczba tab[100];
```

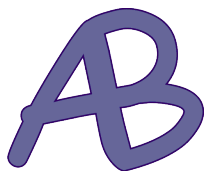
Konstruktor i destruktor

Rozwiązaniem jest utworzenie **tablicy wskaźników do obiektów**, a następnie dynamiczne tworzenie samych obiektów (w pętli)



```
Liczba *tab[100]; // tablica wskaznikow do obiektow
```

```
for (int i = 0; i < 100; i++)  
{  
    tab[i] = new Liczba(rand() % 101);  
}
```



Przesłanie elementów klas

Wskaźnik „this”

Przestanianie nazw

Ponieważ nazwy składników klasy (danych i funkcji) mają zakres klasy, więc w obrębie klasy zastępują elementy o takiej samej nazwie leżące poza klasą.

Np. zmienna `int ile;` będąca składnikiem klasy zastępuje w klasie ewentualną zmienną `ile` o zakresie globalnym lub lokalnym.

```
class Klasa
{
    public:
        int x;
};
int x;
int main()
{
    x=10;
    Klasa ob1;
    ob1.x = 100;
    cout << "x z klasy" << ob1.x << "globalny x" << x;
}
```

Wskaźnik *this*

W niestatycznych metodach klasy występuje wskaźnik **this** - wskazuje on na obiekt, dla którego została wywołana metoda.

```
class Klasa
{
public:
    void metoda()
    {
        std::cout << "Moj adres to " << this << std::endl;
    }
};
```

```
class Klasa
{
    int x;
public:
    void m( int x )
    {
        this->x = x; //Do pola x z klasy przypisujemy argument x
    }
};
```

Przestanianie nazw - przykład

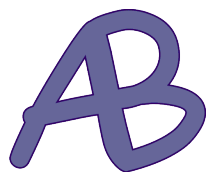
```
using namespace std;

int x=0;

class test
{
public:
    test(int x)
    {
        this -> x=x;
    }
    void wypisz()
    {
        int x=10;
        cout <<"x - lokalny: "<<x<<endl;
        cout <<"x - klasy: "<<this ->x<<endl;
        cout <<"x - globalny programu: "<< ::x<<endl;
    }
private:
    int x;
};

int main()
{
    test t1(20);
    t1.wypisz();
    return 0;
}
```

Operator rozróżniania przestrzeni nazw `::` pozwala dostać się do globalnej zmiennej programu, nawet jeżeli jest przesłonięta przez zmienną obiektu

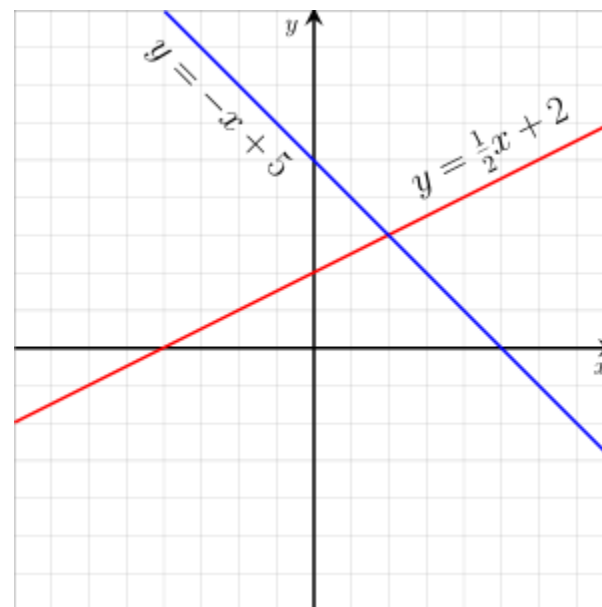


Przykład

Klasa rozwiązująca
równanie liniowe

$$ax + b = 0$$

$$x = \frac{-b}{a}$$



Źródło: Wikipedia


```

#include <iostream>
using namespace std;
//klasa służy do rozwiązywania
//równan liniowych  $ax + b = 0$ 
class rownanieLiniowe
{
    private:
        double a;
        double b;
        double x=0; //miejsce zerowe
        bool rozwiazywalne; //flaga informujaca,
            //czy rownanie ma rozwiazanie
        void wylicz();

    public:
        //konstruktor
        rownanieLiniowe( double a, double b)
        {
            void setA(double a)
            {
                void setB(double b)
                {
                    double getA() {return a;}
                    double getB() {return b;}
                    double getX()
                    {
                        bool czyRozwiazywalne() {return rozwiazywalne;}
                    }
                }
            }
        };

        void rownanieLiniowe::wylicz()
        {

int main()
{

```

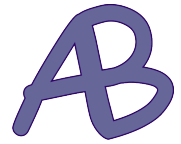


Przykład

Przykład klasy wyliczającej miejsce zerowe równania liniowego $bx+b=0$

Na rysunku widzimy strukturę programu (po zwinięciu treści funkcji)

Treści funkcji zamieszczone na kolejnych stronach (zwróć uwagę na numerację wierszy)



```
public:
    //konstruktor
    rownanieLiniowe( double a, double b)
    {
        this->a=a;
        //"this.a" - to poe klasy
        // "a" - parametr konsstruktor
        this->b=b;
        //po wprowadzeniu danych wyliczymy
        //miejsce zerowe funkcji
        wylicz();
    }
```

Konstruktor z parametrami.
Zwróćmy uwagę na to, że konstruktor automatycznie wywołuje metodę **wylicz()**, która wylicza i oraz sprawdza rozwiązywalność równania

Metoda **setA()** pozwala nadać nową wartość parametrowi **a**. Musi jednak ponownie wywołać metodę **wylicz**, która zaktualizuje **x**

Metoda **setB()**
działa analogicznie

```
void setA(double a)
{
    this->a=a;
    wylicz();
    //po aktualizacji danych ponownie
    //obliczamy miejsce zerowe funkcj
}
void setB(double b)
{
    this->b=b;
    wylicz();
}
```

Przykład c.d.

```
double getA() {return a;}
double getB() {return b;}
double getX()
{
    if (rozwiazywalne)
        return x;
    else
        return 0;
    //jeżeli funkcja nie jest rozwiązywalna
    //zwracamy wartosc 0;
}
bool czyRozwiazywalne() {return rozwiazywalne;}
};
```

Metoda **getX()** zwraca wynik (jeżeli równanie nie jest rozwiązywalne zwróci wartość 0 (nie ma prostej możliwości, aby metoda w takiej sytuacji nie zwróciła wartość) – rozwiązanie tego problemu poznacie na kolejnych wykładach.

Przykład c.d.

```
void rownanieLiniowe::wylicz()  
{  
    if (a==0)  
    {  
        rozwiazywalne=false;  
        return;  
    }  
    else rozwiazywalne=true;  
    //sprawdzamy, czy rownanie mozna rozwiazac  
    x = -b/a;  
}
```

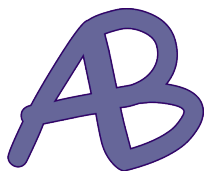
Metoda sprawdza rozwiązywalność i wylicza x;

Zwróćmy uwagę, że treść funkcji opisana jest poza ciałem klasy.

Przykład c.d.

```
int main()
{
    double a,b;
    cout<<"a=";
    cin>>a;
    cout<<"b=";
    cin>>b;
    rownanieLiniowe r1(a,b);
    //tworze instanche klasy "rownanieLiniowe"
    if (r1.czyRozwiazywalne())
        cout<< "Miejsce zerowe:"<<r1.getX();
    else
        cout<<"brak rozwiazan";
    return 0;
}
```

Przykład wykorzystani klasy
rownanieLiniowe



Laboratoria: Zadanie do samodzielnego wykonania

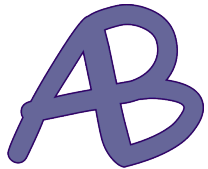
Zadanie 1:

Klasę „Random” przedstawioną na filmie: <https://youtu.be/Rr3qjNS3H0g>

Należy rozbudować o następujące funkcje:

1. Możliwość losowania litery.
2. Możliwość losowania cyfry (zwracanej przez metodę jako znak nie liczba).
3. Losowanie hasła (cyfry i litery). Długość hasła podana w parametrze.
4. Losowanie pinu (cyfry). Długość hasła podana w parametrze. Wywołanie bez parametru oznacza pin 4-ro cyfrowy





Laboratoria: Zadanie do samodzielnego wykonania

Zadanie 2:

Stworzyć klasę losującą zestawy liczb do gier losowych.

Klasa ma losować n liczb ze zbioru od 1 do k .

Klasa powinna obsługiwać 2 wersje losowania:

- z powtórzeniami,
- bez powtórzeń.



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