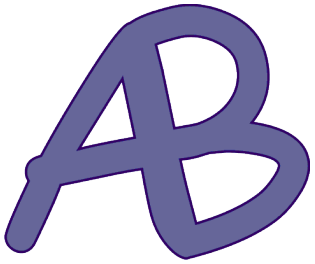
Wykład 4: Klasy cz. 3



- ✓ Konstruktor i destruktor
- ✓ Przesłanianie składników klas
- ✓ Wskaźnik ***this***
- ✓ Przeciążanie metod

Konstruktor i destruktor
(część 1)

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

Konstruktor i destruktor

Dla przykładu przeanalizujmy zainicjalizowanie pola obiektu jakąś wartością.

```
1  #include <iostream>
2  using namespace std;
3
4  class numer {
5    int   liczba ;
6  public:
7    void schowaj(int x)   { liczba = x ; }
8    int zwracaj()   { return liczba ; }
9  } ;
10
11 int main()
12 {
13    numer skrytka;
14    skrytka.schowaj(10);
15    cout << "w skrytce jest liczba " << skrytka.zwracaj() <<endl;
16    return 0;
17 }
```

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***)

Metodą uproszczenia procesu tworzenia obiektu jest dodanie do klasy konstruktora

Konstruktor i destruktor

Tym razem dodamy do obiektu konstruktor

```
1  #include <iostream>
2  using namespace std;
3
4  class numer {
5    int   liczba ;
6  public:
7        numer (int x) {liczba = x;}
8    void schowaj(int x)   { liczba = x ; }
9    int zwracaj()   { return liczba ; }
10 } ;
11
12 int main()
13 {
14   numer skrytka(10);
15   cout << "w skrytce jest liczba " << skrytka.zwracaj() <<endl;
16   return 0;
17 }
```

konstruktor - metoda o takiej samej nazwie jak klasa, uruchamiana automatyczne 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

```
5   class klasa
6   {
7       int pole;
8     public:
9       klasa()   //konstruktor
10      {
11        pole = 10;
12      }
13  };
```

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

```
 5  class klasa
 6  {
 7     public:
 8      int pole;
 9    public:
10      klasa(int wartosc)
11      {             //konstruktor
12        pole = wartosc;
13      }
14  };
15
16  int main()
17  {
18      klasa obiekt01(101);
19      return 0;
20  }
```

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

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

```
18      klasa obiekt01;
```

Takie wywołanie jest w tej sytuacji błędne.

Konstruktor i destruktor

Konstruktor z parametrami

```
 5  class klasa
 6  {
 7     public:
 8      int pole;
 9    public:
10      klasa(int wartosc)
11      {             //konstruktor
12        pole = wartosc;
13      }
14  };
15
16  int main()
17  {
18      klasa obiekt01(101);
19      return 0;
20  }
```

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

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

```
18      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 ~ (wężyk). Podobnie jak konstruktor - nie ma on określenia typu zwracanego.

Konstruktor i destruktor


```
1  #include <iostream>
2  using namespace std;
3
4  class numer {
5    int   liczba ;
6  public:
7          numer (int x) {liczba = x;}
8    void schowaj(int x)   { liczba = x ; }
9    int zwracaj()   { return liczba ; }
10     ~numer () {cout << "no to pa!";}
11 } ;
12
13 int main()
14 {
15     numer skrytka(10);
16     cout << "w skrytce jest liczba " << skrytka.zwracaj() <<endl;
17     return 0;
18 }
```

Destruktor wywołany zostanie przed zamknieciem programu, pomimio, że nie istnieje jego jawne wywołanie

Przesłanianie elementów klas

Wskaźnik „this”

Przesłanianie nazw

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

Np. zmienna int ile;

będąca składnikiem klasy zasłania w klasie ewentualną zmienną ile o zakresie globalnym lub lokalnym.

```
1   class Klasa
2   {
3     public:
4         int x;
5   };
6   int x;
7   int main()
8   {
9       x=10;
10      Klasa ob1;
11      ob1.x = 100;
12      cout << "x z klasy" << ob1.x << "globalny x" << x;
13  }
```

Wskaźnik *this*

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

```
1  class Klasa
2  {
3  public:
4      void metoda()
5      {
6          std::cout << "Moj adres to " << this << std::endl;
7      }
8  };
```

```
1  class Klasa
2  {
3      int x;
4  public:
5      void m( int x )
6      {
7          this->x = x; //Do pola x z klasy przypisujemy argument x
8      }
9  };
```

Przesłanianie nazw - przykład

```
1  #include <iostream>
2  using namespace std;
3
4  int x=0;
5
6  class test
7  {
8  public:
9      test(int x)
10     {
11         this -> x=x;
12     }
13     void wypisz()
14     {
15         int x=10;
16         cout <<"x - lokalny: "<<x<<endl;
17         cout <<"x - klasy: "<<this ->x<<endl;
18         cout <<"x - globalny programu: "<< ::x<<endl;
19     }
20 private:
21     int x;
22 };
23
24 int main()
25 {
26     test t1(20);
27     t1.wypisz();
28     return 0;
29 }
```

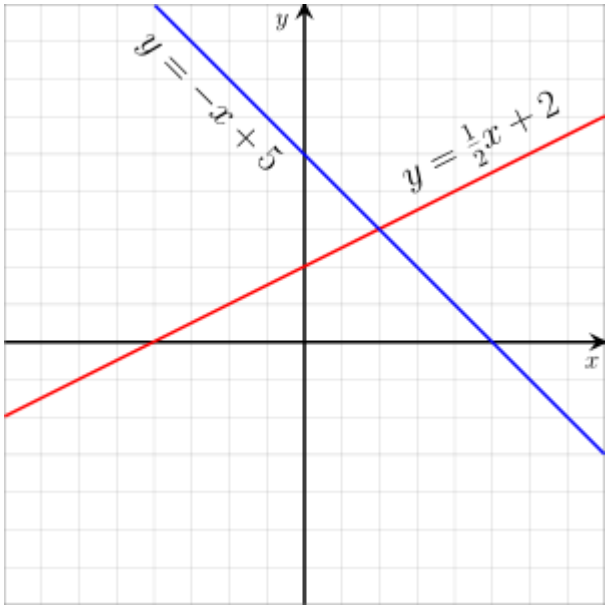
Operator rozróżniania przestrzeni nazw **::** pozwala dostać się do globalnej zmiennej programu, nawet jeżeli jest przesłonięta przez zmienną obiektu

Przykład

Klasa rozwiązująca równanie liniowe

$$ax + b = 0$$

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



Źródło: Wkipedia

```
1  #include <iostream>
2  using namespace std;
3  //klasa służy do rozwiązaywania
4  //równan liniowych ax + b = 0
5  class rownanieLiniowe
6  {
7    private:
8     double a;
9     double b;
10    double x=0; //miejsce zerowe
11    bool rozwiazywalne; //flaga informujaca,
12              //czy rownanie ma rozwiazanie
13    void wylicz();
14
15   public:
16     //konstruktor
17     rownanieLiniowe( double a, double b)
18     {
27     void setA(double a)
28     {
34     void setB(double b)
35     {
39     double getA() {return a;}
40     double getB() {return b;}
41     double getX()
42     {
50     bool czyRozwiazywalne() {return rozwiazywalne;}
51
52 };
53
54 void rownanieLiniowe::wylicz()
55 {
65
66 int main()
67 {
```

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)

Przykład c.d.

```
15      public:
16        //konstruktor
17        rownanieLiniowe( double a, double b)
18        {
19            this->a=a;
20            //"this.a" - to poe klasy
21            // "a"  - parametr konsstruktora
22            this->b=b;
23            //po wprowadzeniu danych wyliczmy
24            //miejsce zerowe funkcji
25            wylicz();
26        }
```

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

```
27      void setA(double a)
28      {
29          this->a=a;
30          wylicz();
31          //po aktualizacji danych ponownie
32          //obliczamy miejsce zerowe funkcj
33      }
34      void setB(double b)
35      {
36          this->b=b;
37          wylicz();
38      }
```

Przykład c.d.

```
39      double getA() {return a;}
40      double getB() {return b;}
41      double getX()
42      {
43          if (rozwiazywalne)
44              return x;
45          else
46              return 0;
47      //jeżeli funkcja nie jest rozwiazywalna
48      //zwracamy wartosc 0;
49      }
50      bool czyRozwiazywalne() {return rozwiazywalne;}
51
52  };
```

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.

```
54  void rownanieLiniowe::wylicz()
55  {
56      if (a==0)
57      {
58          rozwiazywalne=false;
59          return;
60      }
61      else rozwiazywalne=true;
62  //sprawdzamy, czy rownanie mozna rozwiazac
63      x = -b/a;
64  }
```

Metoda sprawdza rozwiązywalność i wylicza x;

Zwróćmy uwagę, ze treść funkcji opisana jest poza ciałem lasy.

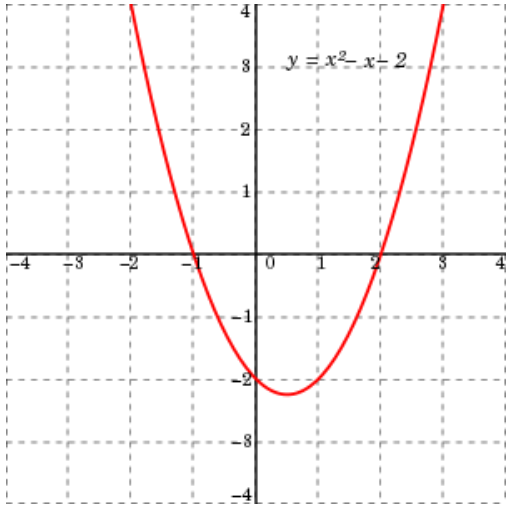
Przykład c.d.

```
66  int main()
67  {
68      double a,b;
69      cout<<"a=";
70      cin>>a;
71      cout<<"b=";
72      cin>>b;
73      rownanieLiniowe r1(a,b);
74      //tworze instanche klasy "rownanieLiniowe"
75      if (r1.czyRozwiazywalne())
76          cout<< "Miejsce zerowe:"<<r1.getX();
77      else
78          cout<<"brak rozwiazan";
79      return 0;
80  }
```

Przykład wykorzystani klasy
rownanieLiniowe

Laboratoria:
Zadanie do samodzielnego wykonania

$$ax^2 + bx + c = 0$$



Źródło: Wkipedia

Zadanie:

Należy napisać klasę rozwiązującą równanie kwadratowe.

Budowa klasy i sposób jej działania powinny być analogiczne do przedstawionej na przykładzie klasy rozwiązującej równanie liniowe.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszj 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.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

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