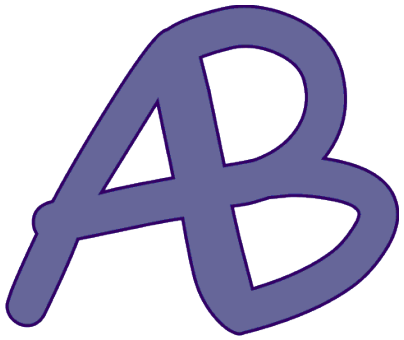


dr Artur Bartoszewski
Katedra Informatyki
UTH Radom

Konstruktor kopiujący

Konstruktor kopiujący

Konstruktor kopiujący to konstruktor, który może zostać wywoływany przez kompilator (niejawnie) jeżeli zachodzi potrzeba stworzenia drugiego egzemplarza obiektu.

Np. podczas przekazywania obiektu do funkcji przez wartość, lub podczas tworzenia nowego obiektu „identycznego” jak już istniejący

- Każda klasa ma konstruktor kopiujący – jeżeli programista go nie napisze, to zostanie on automatycznie utworzony przez kompilator.

```
klasa wzorzec;

        klasa obj1(wzorzec);

        klasa obj2 = wzorzec;

        klasa obj3 = klasa(wzorzec)
```

Konstruktor kopiujący

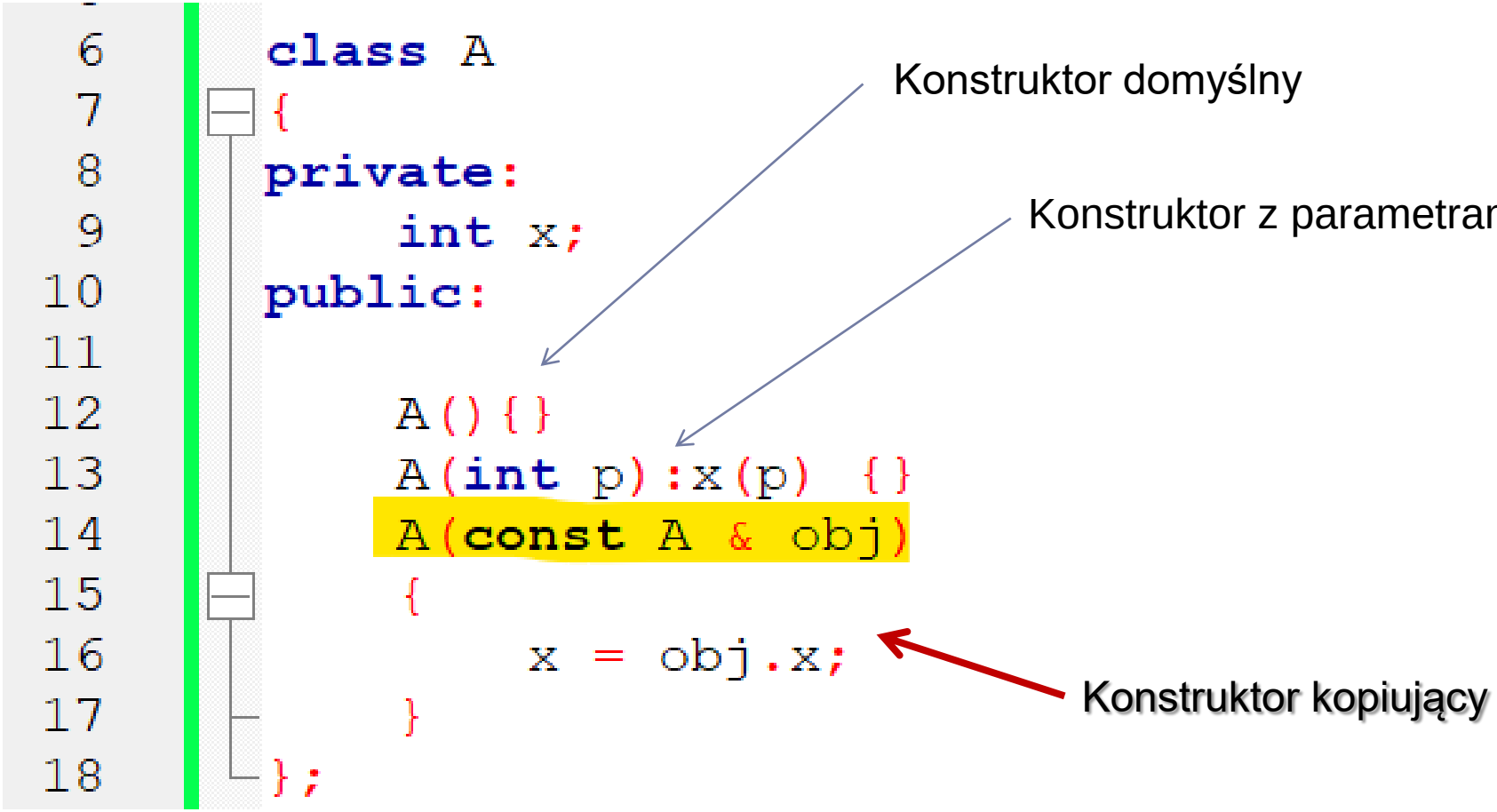
Konstruktor kopiujący przyjmuje referencję do swojego typu i tworzy identyczny obiekt.

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

Konstruktor kopiujący może mieć też inne argumenty, ale musza one mieć wartości domyślne.

```
klasa :: klasa ( klasa & obj , int x=0 , float pi=3.14 , int * p = NULL);
```

Konstruktor kopiujący



Konstruktor kopiujący

```
 6  class A
 7  {
 8  public:
 9      int x;
10      string podpis;
11      A()
12      {
13          cout<<"konstruktor domyslny"<<endl;
14          podpis="obiekt pusty";
15      }
16      A(int px):x(px)
17      {
18          cout<<"konstruktor z parametrami"<<endl;
19          podpis="obiekt z danymi";
20      }
21      A(const A & obj)
22      {
23          x = obj.x;
24          cout<<"konstruktor kopiujacy"<<endl;
25          podpis="kopia obiektu";
26      }
27  };
```

Przykład klasy posiadającej konstruktor kopiujący

Konstruktor kopiujący

Przykład wywołania konstruktor kopiującego

```
32  int main()
33  {
34      A a1, a2(10);
35      a1=A(a2);
36
37      cout<<a1.x<<" "<<a1.podpis;
38      return 0;
39  }
```

```
F:\ProgCpp\konstruktorKopiujacy01\bin\Debug\konstruktorKopiujacy01.exe
konstruktor domyslny
konstruktor z parametrami
konstruktor kopiujacy
10 kopia obiektu
Process returned 0 (0x0)   execution time : 0.058 s
Press any key to continue.
```

```
32  int main()
33  {
34      A a2(10);
35      A a1(a2);
36      cout<<a1.x<<" "<<a1.podpis;
37      return 0;
38  }
```

```
F:\ProgCpp\konstruktorKopiujacy01\bin\Debug\konstruktorKopiujacy01.exe
konstruktor z parametrami
konstruktor kopiujacy
10 kopia obiektu
Process returned 0 (0x0)   execution time : 0.055 s
Press any key to continue.
```

Konstruktor kopiujący

Przykład wywołania konstruktor kopiującego c.d.

```
32  int main()
33  {
34      A a2(10);
35      A a1=a2;
36      cout<<a1.x<<" "<<a1.podpis;
37      return 0;
```

```
F:\ProgCpp\konstruktorKopiujacy01\bin\Debug\konstruktorKopiujacy01.exe
konstruktor z parametrami
konstruktor kopiujacy
10 kopia obiektu
Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

Tak nie zadziała!

```
32  int main()
33  {
34      A a1,a2(10);
35      a1=a2;
36      cout<<a1.x<<" "<<a1.podpis;
37      return 0;
38  }
```

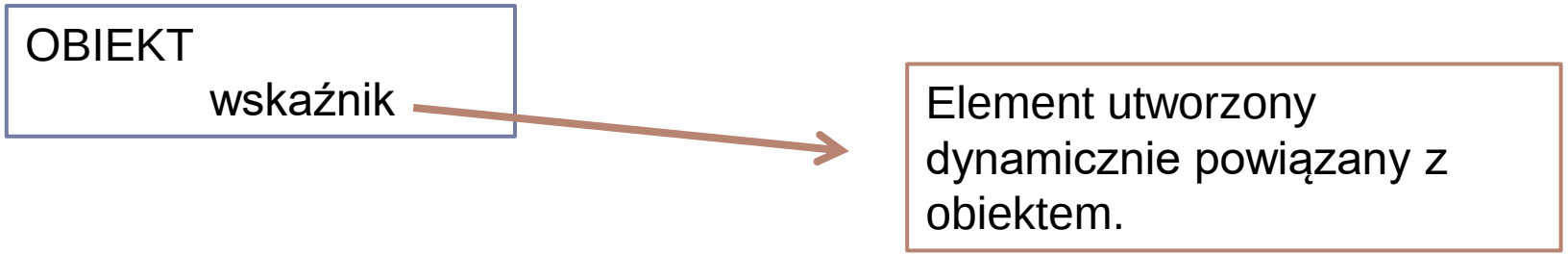
```
F:\ProgCpp\konstruktorKopiujacy01\bin\Debug\konstruktorKopiujacy01.exe
konstruktor domyslny
konstruktor z parametrami
10 obiekt z danymi
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

W ten sposób skopiujemy gotowy obiekt nie wywołując jego konstruktora kopiującego.

Konstruktor kopiujący

Kiedy i dlaczego definiujemy własny konstruktor kopiujący?

Wtedy, gdy składnikami obiektu są wskaźniki na obiekty tworzone dynamicznie.



Konstruktor kopiujący

Przykład: klasa osoba posiada wskaźnik „imie” do zmiennej, którą tworzy dynamicznie

```
 5  class osoba
 6  {
 7  public:
 8      string * imie;
 9      osoba(string kto)
10      {
11          imie = new string;
12          *imie = kto;
13      }
14      ~osoba() {delete imie;}
15      void setImie(string kto){ *imie = kto;}
16      string getimie(){return *imie;}
17  };
```

Konstruktor kopiujący

```
F:\ProgCpp\konstruktorKopiujacy02\bin\Debug\konstruktorKopiujacy02.exe
Kowalski
Nowak
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

```
20  int main()
21  {
22      osoba ktos("Kowalski");
23      osoba ktosInny("Nowak");
24
25      cout<<ktos.getimie()<<endl
26          <<ktosInny.getimie();
27
28      return 0;
```

Z pozoru wszystko działa poprawnie – dopóki nie spróbujemy tworzyć obiektów poprze kopiowanie.

Konstruktor kopiujący

```
20  int main()
21  {
22      osoba ktos("Kowalski");
23      osoba ktosInny = ktos;
24
25      ktos.setImie("Iksinski");
26
27      cout<<ktos.getimie()<<endl
28          <<ktosInny.getimie();
29
30      return 0;
31  }
```

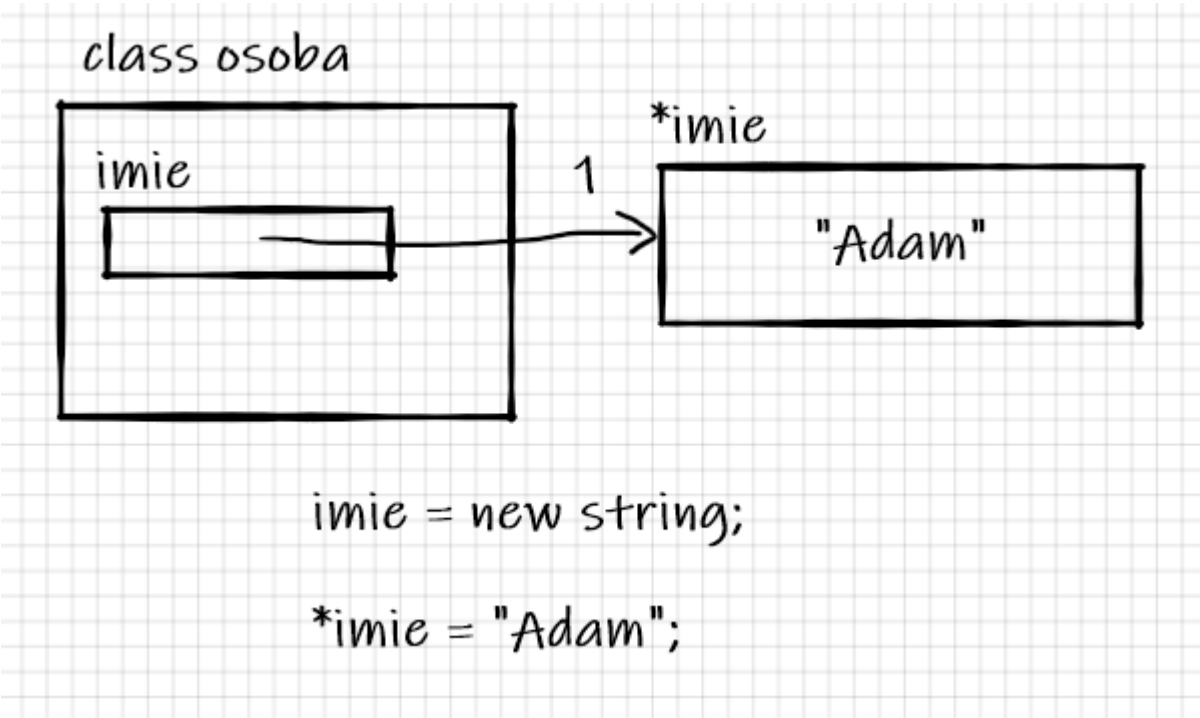
Obiekt *„ktosInny”* utworzyliśmy kopiując obiekt *„ktos”* i dopiero w następnym kroku zmieniliśmy jego zawartość.

Okazuje się jednak, że zmianie uległa zawartość przechowywana przez oba obiekty.

```
F:\ProgCpp\konstruktorKopiujacy02\bin\Debug\konstruktorKopiujacy02.exe
Iksinski
Iksinski
Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

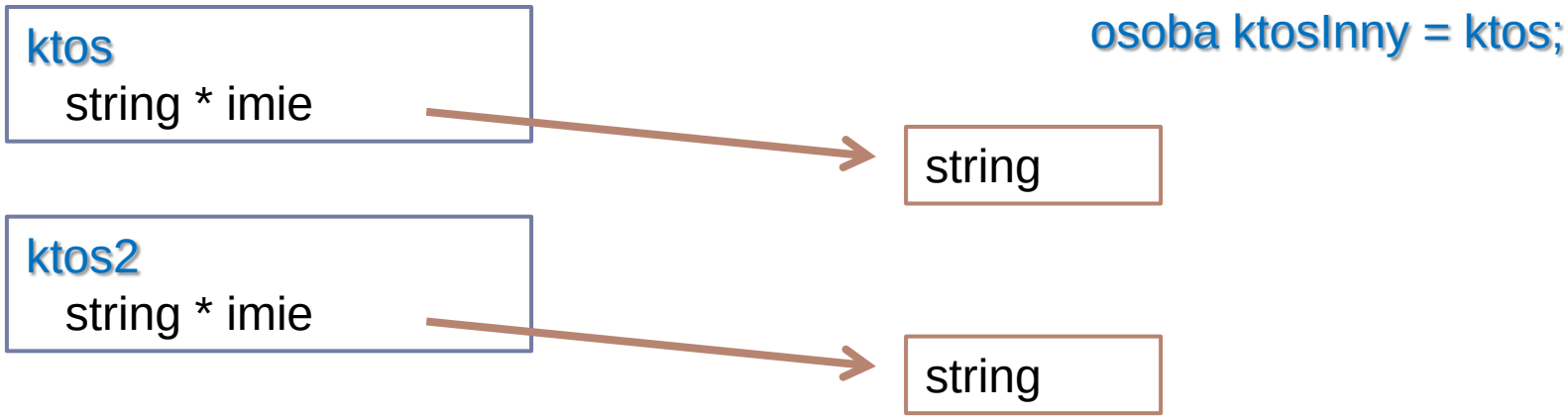
Konstruktor kopiujący

Obiekt klasy ***osoba*** zawiera wskaźnik do zmiennej typu *string* tworzonej dynamicznie w konstruktorze obiektu.



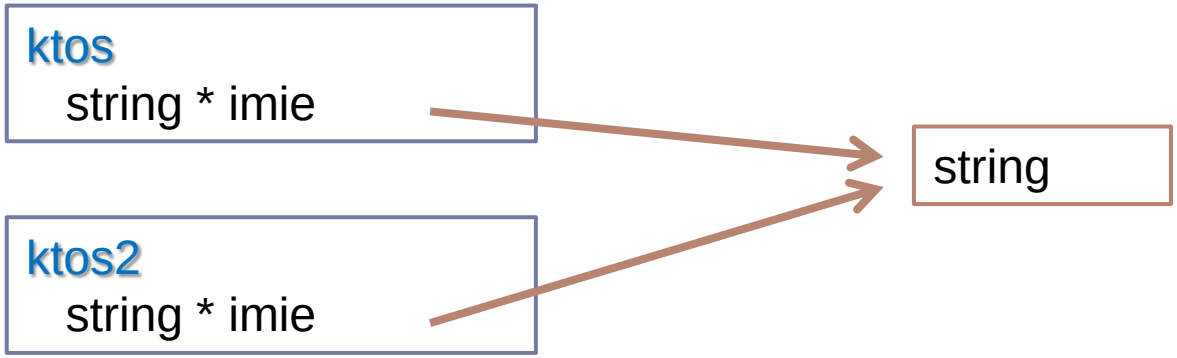
Konstruktor kopiujący

Kopiując obiekt spodziewamy się takiej sytuacji:



Jednak otrzymamy taką:

Wskaźniki *imie* we wszystkich kopiach obiektu zostały dosłownie skopiowane – czyli wskazują na ten sam element. Zmaina w jednym obiekcie pociąga za sobą zmianę we wszystkich



Konstruktor kopiujący

```
5   class osoba
6   {
7   public:
8       string * imie;
9       osoba(string kto)
10      {
11          imie = new string;
12          *imie = kto;
13      }
14      osoba (osoba &);
15      ~osoba() {delete imie;}
16      void setImie(string kto){ *imie = kto;}
17      string getimie(){return *imie;}
18  };
19  osoba::osoba(osoba & wzorzec)
20  {
21      imie = new string;
22      *imie = *wzorzec.imie;
23      //*imie = wzorzec.getimie(); //- inna metoda
24  }
```

Konstruktor kopiujący, który nie skopiuje pola *imie* jeden do jednego, a utworzy nowy łańcuch dynamiczny, przekopiuje do niego dane z wzorca i wstawi jego adres do pola *imie*

Tym razem zadział prawidłowo.

```
F:\ProgCpp\konstruktorKopiujacy02\bin\Debug\konstruktorKopiujacy02.exe
Kowalski
Iksinski
Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

Konstruktor kopiujący

Wywołanie konstruktora kopiującego dla obiektów tworzonych dynamicznie

```
27  int main()
28  {
29      osoba * wzor = new osoba("Kowalski");
30      osoba * ktos = new osoba(*wzor);
31
32      ktos->setImie("Iksinski");
33
34      cout<<wzor->getimie()<<endl
35          <<ktos->getimie();
36
37      return 0;
38  }
```

Konstruktor kopiujący

Przykład: klasa student

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

class student
{
protected:
    int Id;
    string imie;
    string nazwisko;
    string kierunek;
    string wydzial;
    string *historia_studiow;
public:
    static int ile;
    student(string im = "", string nazw = "", string k = "", string w = "") : imie(im), nazwisko(nazw), kierunek(
k), wydzial(w)
    {
        Id = ++ile;
        historia_studiow = new string;
        *historia_studiow = "";
    }
    student(const student &wzorzec, string im = "", string nazw = "")
    {
        Id = ++ile;
        imie = im;
        nazwisko = nazw;
        kierunek = wzorzec.kierunek;
        wydzial = wzorzec.wydzial;
        // historia_studiow = wzorzec.historia_studiow; // - kopiowanie płytkie
        historia_studiow = new string;
        * historia_studiow = "";
    }
```

Konstruktor kopiujący

```
    void addWpis(string tekst)
    {
        string temp = *historia_studiow;
        temp += "\n";
        temp += tekst;
        *historia_studiow = temp;
    }

    string toString()
    {
        stringstream temp;
        temp << "(" << Id << ") " << nazwisko << " " << imie << " Wydzial: " << wydzial << " Kierunek: " <<
 kierunek
             << endl
             << "Historia studiow: " << *historia_studiow;
        return temp.str();
    }

    ~student()
    {
        delete historia_studiow;
    }
};

int student::ile = 0;
```

Konstruktor kopiujący

```
int student::ile = 0;

student kierunki[] = {
    student("", "", "Informatyka techniczna", "WTEiI"),
    student("", "", "Informatyka ogolnoakademicka", "WTEiI"),
    student("", "", "Pedagogika", "WFP"),
    student("", "", "Transport", "WTEiI")};

int main()
{
    student::ile = 0;

    student s1(kierunki[0], "Jan", "Kowalski");
    s1.addWpis("Przyjety na studia.");
    s1.addWpis("Zdal na 2 semestr");

    student s2(kierunki[0], "Andrzej", "Nowak");
    s2.addWpis("Przyjety na studia.");
    s2.addWpis("Skreslony z listy studentow");

    cout << s1.toString()<<endl;
    cout << s2.toString();

    return 0;
}
```

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