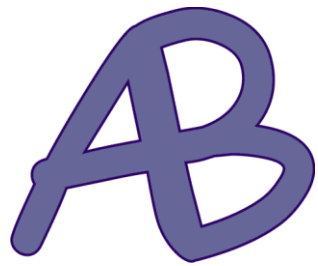


dr Artur Bartoszewski
Katedra Informatyki i teleinformatyki
Uniwersytet Radomski



Wykład:



Metoda .toString()

W języku c# każda klasa posiada metodę .ToString(), która zwraca w postaci tekstowej dane przechowywane przez obiekt, lub też informuje do jakiej klasy należy obiekt jeżeli takich danych nie można sensownie wypisać.

Ponieważ posiadanie metody, która w prosty sposób poinformuje nas jaki jest aktualny stan obiektu może być przydatne w trakcie wykładów nauczymy się taką metodę tworzyć.

Metoda .toString()

```
#include <sstream>
```

```
class Osoba
{
private:
    string imie;
    string nazwisko;
    int wiek;

public:
    string toString()
    {
        stringstream bufor;
        bufor << imie << " " << nazwisko << " " << wiek;
        return bufor.str();
    }
};
```



dr Artur Bartoszewski
Katedra Informatyki i teleinformatyki
Uniwersytet Radomski

Przekazywanie do funkcji argumentów będących obiektami



Przekazywanie obiektów do funkcji

Obiekt jest rozbudowaną wersją zmiennej.

Podobnie jak zmienne typu `int` czy `double`, każdy obiekt można przekazać do funkcji jako argument (mowa tu o funkcji z poza klasy, nie metodzie wchodzącej w skład klasy).

Przekazywanie obiektów do funkcji

Przekazywanie obiektu przez wartość

Domyślnie przesyłamy obiekt przez wartość – czyli na potrzeby funkcji tworzona jest jego kopia (tak samo jak w przypadku zmiennej prostej)

Uwaga: Jeśli obiekt jest duży, to proces kopiowania może trwać dłużej. Wielokrotne wysłanie przez wartość może wyraźnie wpływać na zwolnienie programu.

Przekazywanie obiektów do funkcji

```
class Osoba
{
private:
    string imie;
    string nazwisko;
    int wiek;
public:
    Osoba(string IMIE, string NAZWISKO, int WIEK) : imie(IMIE), nazwisko(NAZWISKO), wiek(WIEK) { }
    void setImie(string imie) { this->imie = imie; }
    void setNazwisko(string nazwisko) { this->nazwisko = nazwisko; }
    void setWiek(int wiek) { this->wiek = wiek; }
    string getImie() { return imie; }
    string getNazwisko() { return nazwisko; }
    int getWiek() { return wiek; }
    string toString()
    {
        stringstream bufor;
        bufor << imie << " " << nazwisko << " " << wiek;
        return bufor.str();
    }
    ~Osoba() { }
};
```

Przykład:

Przekazywanie obiektu przez wartość

Przekazywanie obiektów do funkcji

```
void RODO(Osoba temp);  
int main()  
{  
    Osoba ktos("Jan", "Kowalski", 30);  
    cout << "Przed zmiana:" << ktos.toString() << endl;  
    RODO(ktos);  
    cout << "Po zmianie:" << ktos.toString() << endl;  
    return 0;  
}
```

Obiekt klasy „Osoba”
przekazujemy do funkcji
„rodo()” wewnątrz której
jest modyfikowany

```
void RODO(Osoba temp)  
{  
    string s = temp.getNazwisko();  
    for (int i = 1; i < s.length(); i++)  
        s[i] = '*';  
    temp.setNazwisko(s);  
}
```

Niestety nie
zadziałało....

Przekazywanie obiektów do funkcji

```
void RODO(Osoba temp)
{
    string s = temp.getNazwisko();
    for (int i = 1; i < s.length(); i++)
        s[i] = '*';
    temp.setNazwisko(s);
}
```

Należy pamiętać, że przekazując obiekt przez wartość wysyłamy do funkcji jego kopię, która jest usuwana z pamięci po zakończeniu funkcji.

Zmiany wprowadzone na kopii nie wpłyną na oryginał.

Przekazywanie obiektów do funkcji

Przekazywanie obiektu przez referencję

Referencja jest drugą nazwą, „przezwiskiem” - nie przezwiskiem klasy, ale danego egzemplarza jej obiektu. Wysyłając taki egzemplarz obiektu do funkcji na zasadzie przesłania przez referencję - sprawiamy, że nie jest on kopiowany. **Funkcja ma dostęp do oryginału.**

Przekazywanie obiektów do funkcji

```
class Osoba
{
private:
    string imie;
    string nazwisko;
    int wiek;
public:
    Osoba(string IMIE, string NAZWISKO, int WIEK) : imie(IMIE), nazwisko(NAZWISKO), wiek(WIEK) { }
    void setImie(string imie) { this->imie = imie; }
    void setNazwisko(string nazwisko) { this->nazwisko = nazwisko; }
    void setWiek(int wiek) { this->wiek = wiek; }
    string getImie() { return imie; }
    string getNazwisko() { return nazwisko; }
    int getWiek() { return wiek; }
    string toString()
    {
        stringstream bufor;
        bufor << imie << " " << nazwisko << " " << wiek;
        return bufor.str();
    }
    ~Osoba() { }
};
```

Przekazywanie obiektu przez referencję - przykład

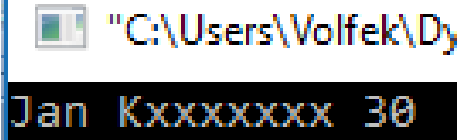
Klasa wygląda dokładnie tak samo jak w poprzednim przykładzie

Przekazywanie obiektów do funkcji

```
void RODO(Osoba &temp);  
int main()  
{  
    Osoba ktos("Jan", "Kowalski", 30);  
    cout << "Przed zmiana:" << ktos.toString() << endl;  
    RODO(ktos);  
    cout << "Po zmianie:" << ktos.toString() << endl;  
    return 0;  
}
```

```
void RODO(Osoba &temp)  
{  
    string s = temp.getNazwisko();  
    for (int i = 1; i < s.length(); i++)  
        s[i] = '*';  
    temp.setNazwisko(s);  
}
```

Przekazujemy obiekt przez referencję – **tym razem zadziała prawidłowo**



```
"C:\Users\Volfek\Desktop"  
Jan Kxxxxxxxx 30
```

Przekazywanie obiektów do funkcji

Wywołanie konstruktora i destruktora obiektu przekazywanego wartość,
przez referencję oraz Przez wskaźnik do obiektu utworzonego
dynamicznie.

Sposób przekazywania obiektu do funkcji wpływa na sposób
uruchamiania się konstruktora i destruktoru obiektu.
Przyjrzyjmy się temu zagadnieniu na przykładach

Przekazywanie obiektów do funkcji

```
class Test
{
public:
    Test() { cout << "Utworzono obiekt" << endl; }
    ~Test() { cout << "Usunieto obiekt" << endl; }
};
void funkcja(Test temp);
```

```
int main()
{
    Test t1;      // tworzenie obiektu
    funkcja(t1);  // kopiowanie - nie jest uruchamiany konstruktor
    return 0;
}
```

```
void funkcja(Test temp)
{
}
```

Obiekt został przez przekazany
przez wartość

W efekcie konstruktor obiektu
uruchomiony został raz a
destruktor 2 razy

```
PS D:\PROGRAMOWANIE\2025\prog_ob> .\a.exe
Utworzono obiekt
Usunieto obiekt
Usunieto obiekt
```

Przekazywanie obiektów do funkcji

```
class Test
{
public:
    Test() { cout << "Utworzono obiekt" << endl; }
    ~Test() { cout << "Usunieto obiekt" << endl; }
};

void funkcja(Test &temp);

int main()
{
    Test t1;      // tworzenie obiektu
    funkcja(t1);  // kopiowanie - nie jest uruchamiany konstruktor
    return 0;
}

void funkcja(Test $temp)
{
}
}
```

Obiekt został przez przekazany
przez referencję

W efekcie konstruktor i
destruktor obiektu
uruchomiony zostały
jednokrotnie

PS D:\PROGRAMOWANIE\2025\prog_ob> .\a.exe
Utworzono obiekt
Usunieto obiekt

Przekazywanie obiektów do funkcji

```
class Test
{
public:
    Test() { cout << "Utworzono obiekt" << endl; }
    ~Test() { cout << "Usunieto obiekt" << endl; }
};

void funkcja(Test *temp);

int main()
{
    Test *t1 = new Test; // tworzenie obiektu - dynamiczne
    funkcja(t1);          // przekazywanie wskaźnika
    delete t1;
    return 0;
}

void funkcja(Test *temp)
{
}
```

Obiekt Utworzony został dynamicznie a do funkcji przekazany został wskaźnik do obiektu

Podobnie jak w poprzednim przykładzie konstruktor i destruktor obiektu uruchomiony zostały jednokrotnie

PS D:\PROGRAMOWANIE\2025\prog_ob> .\a.exe
Utworzono obiekt
Usunieto obiekt

Przekazywanie obiektów do funkcji

Przekazywanie obiektu przez wartość

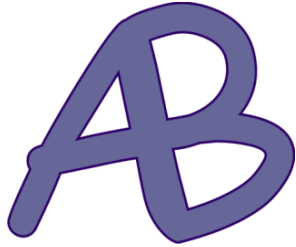
- Przy wywołaniu funkcji następuje kopiowanie obiektu, co oznacza wywołanie konstruktora kopiującego – a takiego konstruktora w naszym przykładzie nie utworzyliśmy. Nauczymy się to robić na jednym z następnych wykładów.
- Po zakończeniu funkcji kopia obiektu jest niszczona, co powoduje wywołanie destruktora.

Przekazywanie obiektu przez referencję (&)

- Nie tworzy się kopia obiektu, więc nie wywołuje się dodatkowy konstruktor.
- Funkcja operuje na tym samym obiekcie, który został przekazany, a destrukcja następuje tylko raz – gdy obiekt wyjdzie z zakresu, w którym został utworzony.

Przekazywanie obiektu przez wskaźnik (*) do obiektu utworzonego dynamicznie (new)

- Przekazujemy adres obiektu, więc nie wywołuje się dodatkowy konstruktor.
- Destrukcja obiektu nie następuje automatycznie – należy jawnie wywołać delete, aby zwolnić pamięć i uruchomić destruktorem.



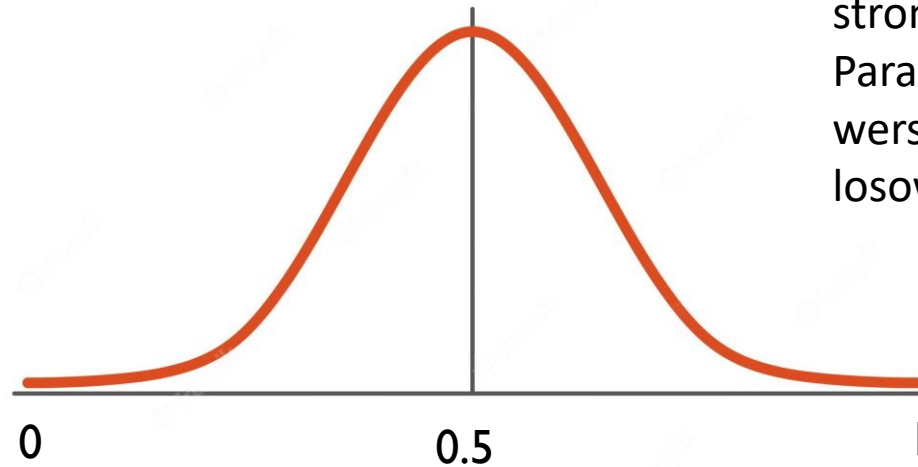
dr Artur Bartoszewski
Katedra Informatyki i teleinformatyki
Uniwersytet Radomski

Mechanizm losujący

Będzie nam potrzebny w następnym wykładzie

Mechanizm losujący:

Dzięki zastąpieniu pojedynczego losowania średnią z wielu losowań uzyskamy rozkład wyników zbliżony do rozkładu normalnego (krzywej Gaussa)



Parametr **skupienie** reguluje to jak stroma jest krzywa.

Parametr **minimum**, w przeciążonej wersji funkcji, zawęży zakres losowania $\langle \text{minimum}/100, 1 \rangle$

Mechanizm losujący:

```
double procentGauss(int skupienie) // <0,1)
{
    double wynik = 0.0;
    for (int i = 0; i < skupienie; i++)
        wynik += rand() % 100;
    wynik /= skupienie;
    return wynik / 100.0;
}
```

```
double procentGauss(int minimum, int skupienie) // <minimum/100 , 1)
{
    double wynik = 0.0;
    for (int i = 0; i < skupienie; i++)
        wynik += rand() % (100 - minimum);
    wynik = wynik / skupienie + minimum;
    return wynik / 100.0;
}
```

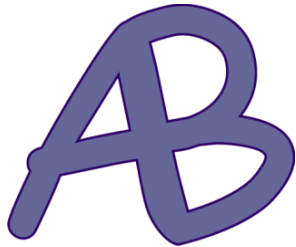
Pomocniczo, nasza klasa będzie posiadała 2 metody służące do losowania.

Mechanizm losujący:

```
double procentGauss(int skupienie) // <0,1)
{
    double wynik = 0.0;
    for (int i = 0; i < skupienie; i++)
        wynik += rand() % 100;
    wynik /= skupienie;
    return wynik / 100.0;
}
```

```
double procentGauss(int minimum, int skupienie) // <minimum/100 , 1)
{
    double wynik = 0.0;
    for (int i = 0; i < skupienie; i++)
        wynik += rand() % (100 - minimum);
    wynik = wynik / skupienie + minimum;
    return wynik / 100.0;
}
```

Pomocniczo, nasza klasa będzie posiadała 2 metody służące do losowania.



dr Artur Bartoszewski
Katedra Informatyki i teleinformatyki
Uniwersytet Radomski

Przykład – powtórzenie dotychczasowych wiadomości

Przykład:

```
#include <iostream>
#include <cmath>
#include <sstream>

using namespace std;

class RownanieKwadratowe
{
private:
    double a, b, c;
    double delta, x1, x2;

    void liczDelta()
    {
        delta = b * b - 4 * a * c;
    }
}
```

- Nazwa klasy

- Nazwy metod

- Wywołania przez metody, innych metod w łasnej klasy

```
void liczPierwiastki()
```

```
{
    if (a == 0)
    {
        delta = 0;
        x1 = x2 = NAN; // Nie można dzielić przez zero
    }
    else
    {
        liczDelta();
        if (delta > 0)
        {
            x1 = (-b - sqrt(delta)) / (2 * a);
            x2 = (-b + sqrt(delta)) / (2 * a);
        }
        else if (delta == 0)
        {
            x1 = -b / (2 * a);
            x2 = x1;
        }
        else
        {
            x1 = x2 = NAN; // Brak rozwiązań rzeczywistych
        }
    }
}
```

Co to jest NAN?
Objaśnienie na następnym slajdzie

STAŁA NAN:

Stała NAN (ang. Not a Number) to specjalna wartość typu double, oznaczająca wynik nieokreślony lub nieprawidłowy w operacjach matematycznych.

- ✓ Reprezentuje niezdefiniowane wyniki, np. $0.0 / 0.0$ lub $\text{sqrt}(-1)$.
- ✓ Można jej używać do sygnalizowania błędów w obliczeniach.
- ✓ NAN nie jest równe nawet samemu sobie – porównanie `NAN == NAN` zwraca false.

Przykład:

```
public:
    RownanieKwadratowe(double A, double B, double C) :
    a(A), b(B), c(C), delta(0), x1(0), x2(0)
    {
        liczPierwiastki();
    }
```

```
string toString()
{
    stringstream odp;
    if (a == 0)
    {
        odp << "To nie jest równanie kwadratowe.\n";
    }
    else if (delta > 0)
    {
        odp << "x1 = " << x1 << endl
            << "x2 = " << x2 << endl;
    }
    else if (delta == 0)
    {
        odp << "x0 = " << x1 << endl;
    }
    else
    {
        odp << "Brak rozwiązań w zbiorze liczb rzeczywistych.\n";
    }
    return odp.str();
}
```

Przykład:

```
double getDelta() { return delta; }
double getX1() { return x1; }
double getX2() { return x2; }
bool rozwiazania() { return delta >= 0; }

void setA(double A)
{
    a = A;
    liczPierwiastki();
}
void setB(double B)
{
    b = B;
    liczPierwiastki();
}
void setC(double C)
{
    c = C;
    liczPierwiastki();
}
};
```

```
int main()
{
    RownanieKwadratowe r1(3, 6, -7);
    cout << r1.toString();

    if (r1.rozwiazania())
    {
        cout << "Rozwiązania: " << r1.getX1()
              << " " << r1.getX2()
              << endl;
    }

    return 0;
}
```

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