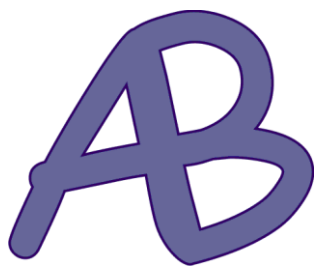


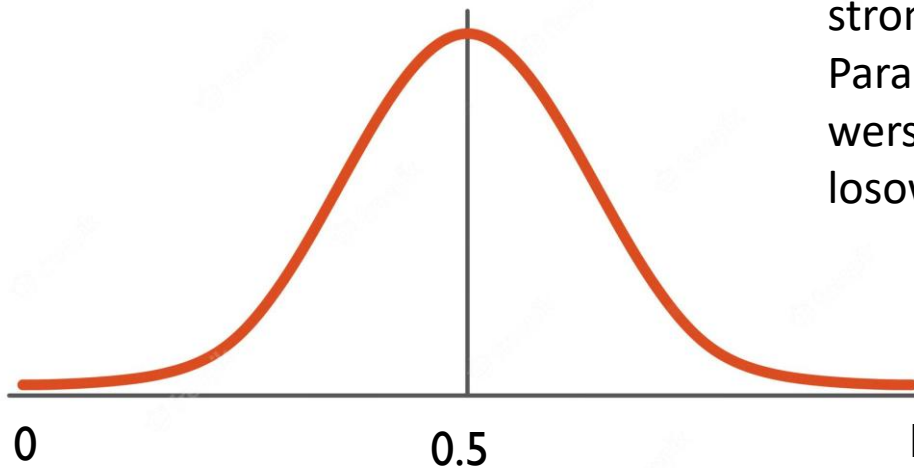
*dr Artur Bartoszewski*  
*Katedra Informatyki i teleinformatyki*  
*Uniwersytet Radomski*



## **Wykład: Przykład praktyczny**

## Mechanizm losujący:

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



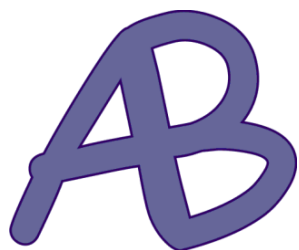
Parametr **skupienie** reguluje to jak stroma jest krzywa.  
Parametr **minimum**, w przeciążonej wersji funkcji, zawęży zakres losowania  $\langle \text{minimum}/100, 1 \rangle$

## 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.



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

## **Przykład: gra RPG**

### **Wersja rozwojowa**



# Przykład gra RPG v 0.1

Tworzymy bardzo prostą tekstową grę RPG, w której będziemy wystawiać dwa obiekty klasy „postać” do walki w funkcji „ring”

Niestety, grafiki jednak nie będzie...

## Przekazywanie obiektów do funkcji - przykład

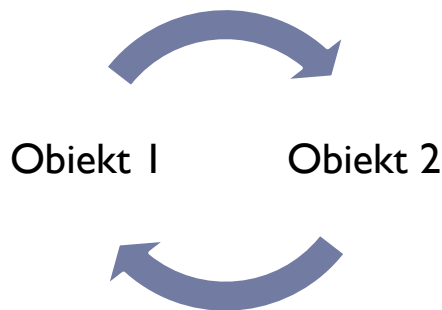
```
const int ZYCIE = 100;  
const int PUNKTY = 30;
```

Stałe życie i punkty posłużą do losowania atrybutów postaci

```
struct cios  
{  
    int atak_fizyczny = 0;  
    int atak_magiczny = 0;  
    bool podstepny_atak;  
};
```

Struktura cios opisuje parametry zadanego przez postacie ciosu składa się ona z dwóch komponentów:

- ataku fizycznego,
  - ataku magicznego
- oraz informacji czy wykonany atak był „podstępny”



## Przekazywanie obiektów do funkcji - przykład

```
class postac
{
private:
    string imie;
    int zycie;
    int atak_fizyczny;
    int atak_magiczny;
    int obrona_fizyczna;
    int obrona_magiczna;

public:
    postac(string im) : imie(im){}
    string przedstaw_sie() {}
    bool czy_zyje() {}
    cios zadaj_cios() {}
    bool przyjmij_cios(cios c) {} // true - zycje, false - nie zyje
    string getImie() { return imie; }
};
```

### Budowa klasy postać

- pola opisują parametry postaci
- konstruktor posłuży do wylosowania tych parametrów
- pozostałe metody będą użyte podczas walki.

Zwróćmy uwagę na wykorzystanie struktury cios opisanej na poprzednim slajdzie

## Przekazywanie obiektów do funkcji - przykład

Konstruktor otrzymuje w parametrze imię postaci

Atak to wylosowany procent ze stałej PUNKTY

Obrona jest odwrotnie proporcjonalna do ataku

```
postac(string im) : imie(im)
{
    zycie = ZYCIE;
    int atak = round(PUNKTY * procentGauss(3));
    int obrona = PUNKTY - atak;
    atak_fizyczny = round(atak * procentGauss(3));
    atak_magiczny = atak - atak_fizyczny;
    obrona_fizyczna = round(obrona * procentGauss(3));
    obrona_magiczna = obrona - obrona_fizyczna;
}
```

A tak rozdzielany jest losowo na 2 składowe:

- atak fizyczny,
- atak magiczny.

Podobnie wyliczana jest obrona fizyczna i magiczna



## Przekazywanie obiektów do funkcji - przykład

```
string przedstaw_sie()  
{  
    stringstream w;  
    w << "\nImie: " << imie  
      << " Zycie: " << zycie << endl  
      << "Atak fizyczny: " << atak_fizyczny  
      << " Atak magiczny: " << atak_magiczny << endl  
      << "Obrona fizyczna: " << obrona_fizyczna  
      << " Obrona magiczna: " << obrona_magiczna << endl;  
    return w.str();  
}
```

Tworzymy strumień tekstowy „w” i dodajemy do niego opisy i wartości pól klasy

Konwertujemy strumień tekstowy na zwykły string który zwracamy jako wartość funkcji

## Przekazywanie obiektów do funkcji - przykład

---

```
bool czy_zyje()  
{  
    if (zycie > 0)  
        return true;  
    else  
        return false;  
}
```

Funkcja uprości sprawdzanie  
czy walka się skończyła

## Przekazywanie obiektów do funkcji - przykład

Metoda „zadaj\_cios” losuje wartości składowych ciosu.

Na ich podstawie tworzy strukturę „c” typu „cios” którą zwraca w wartości funkcji.

```
cios zadaj_cios()
{
    cios c;
    c.atak_fizyczny = round(atak_fizyczny * procentGauss(20, 3));
    c.atak_fizyczny = round(atak_fizyczny * procentGauss(20, 3));
    c.atak_magiczny = round(atak_magiczny * procentGauss(20, 3));
    if (rand() % 5 == 0)
        c.podstepny_atak = true;
    else
        c.podstepny_atak = false;

    return c;
}
```

## Przekazywanie obiektów do funkcji - przykład

```
bool przyjmij_cios(cios c) // true - zycje, false - nie zyje
{
    int obrazenia_fizyczne;
    int obrazenia_magiczne;
    if (c.podstepny_atak)
    {
        obrazenia_fizyczne = c.atak_fizyczny;
        obrazenia_magiczne = c.atak_magiczny;
    }
    else
    {
        obrazenia_fizyczne = c.atak_fizyczny - round(obrona_fizyczna * procentGauss(2));
        obrazenia_magiczne = c.atak_magiczny - round(obrona_magiczna * procentGauss(2));
    }
    if (obrazenia_fizyczne < 0)
        obrazenia_fizyczne = 0;
    if (obrazenia_magiczne < 0)
        obrazenia_magiczne = 0;
    zycie = zycie - (obrazenia_fizyczne + obrazenia_magiczne);

    if (zycie > 0)
        return true;
    else
        return false;
}
```

Na podstawie struktury cios oraz obrony postaci wyliczone będą obrażenia które odniosła nasza postać.

Podstępny atak powoduje, że obrona postaci nie jest uwzględniana

Należy zabezpieczyć się przed tym, żeby ujemne obrażenia (możliwe gdy obrona jest większa od siły ciosu atakującego) nie zaczęły leczyć atakowanej postaci.

Życie postaci zmniejszamy o otrzymane obrażenia i sprawdzamy czy nie spadło ono do zera.

## Przekazywanie obiektów do funkcji - przykład

```
void ring(postac zawodnik_1, postac zawodnik_2)
{
    cios c;
    int max_liczba_rund = 100;

    cout << zawodnik_1.przedstaw_sie();
    cout << zawodnik_2.przedstaw_sie();

    while (zawodnik_1.czy_zyje() && zawodnik_2.czy_zyje() && --max_liczba_rund > 0)
    {
        c = zawodnik_1.zadaj_cios();
        cout << endl
              << zawodnik_1.getImie() << ": "
              << c.atak_fizyczny << "/" << c.atak_magiczny;
        zawodnik_2.przyjmij_cios(c);
        c = zawodnik_2.zadaj_cios();
        cout << endl
              << zawodnik_2.getImie() << ": "
              << c.atak_fizyczny << "/" << c.atak_magiczny;
        zawodnik_1.przyjmij_cios(c);
    }
    cout << zawodnik_1.przedstaw_sie();
    cout << zawodnik_2.przedstaw_sie();
}
```

Funkcja ring nie jest częścią składową klasy.

Do funkcji przesyłane są (w tym wypadku przez wartość) dwa obiekty klasy postać

Wewnątrz pętli następuje wymiana ciosów. Zawodnik\_1 generuje strukturę cios, a zawodnik\_2 ją przyjmuje. Następnie operacja powtarzana jest w przeciwną stronę. Zakładamy, że przeciwnik zawsze „odda” nawet jeśli jego życie spadło poniżej zera. Inaczej atakujący miałby przewagę

## Przekazywanie obiektów do funkcji - przykład

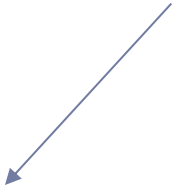
```
void ring(postac zawodnik_1, postac zawodnik_2)
{
    cios c;
    int max_liczba_rund = 100;

    cout << zawodnik_1.przedstaw_sie();
    cout << zawodnik_2.przedstaw_sie();

    while (zawodnik_1.czy_zyje() && zawodnik_2.czy_zyje() && --max_liczba_rund > 0)
    {
        c = zawodnik_1.zadaj_cios();
        cout << endl
             << c.atak_fizyczny << " " << c.atak_magiczny;
        zawodnik_2.przyjmij_cios(c);
        c = zawodnik_2.zadaj_cios();
        cout << endl
             << c.atak_fizyczny << " " << c.atak_magiczny;
        zawodnik_1.przyjmij_cios(c);
    }

    cout << zawodnik_1.przedstaw_sie();
    cout << zawodnik_2.przedstaw_sie();
}
```

Pętla powtarzana jest dopóki życie obu zawodników jest powyżej zera. Walka zostanie też zatrzymana po 100 rundach



## Przekazywanie obiektów do funkcji - przykład

W funkcji **main()** tworzymy dwie postacie i wysyłamy je do funkcji **ring** - o resztę zadba klasa

```
int main()
{
    srand(time(NULL));
    postac p1("Conan"), p2("Rambo");
    ring(p1, p2);
    return 0;
}
```

Efekt działania programu:

```
Imie: Conan Zycie: 100
Atak fizyczny: 5 Atak magiczny: 14
Obrona fizyczna: 6 Obrona magiczna: 5
```

```
Imie: Rambo Zycie: 100
Atak fizyczny: 8 Atak magiczny: 12
Obrona fizyczna: 5 Obrona magiczna: 5
```

```
Conan: 3/9
Rambo: 4/6
Conan: 3/8
Rambo: 5/6
Conan: 4/11
Rambo: 6/4
Conan: 2/13
Rambo: 6/6
Conan: 4/8
Rambo: 5/6
Conan: 4/9
Rambo: 4/8
Conan: 4/6
Rambo: 5/6
Conan: 2/7
Rambo: 7/6
Conan: 3/9
Rambo: 5/6
Conan: 3/9
Rambo: 7/5
Conan: 4/9
Rambo: 6/6
Conan: 2/7
Rambo: 5/5
Conan: 3/9
Rambo: 4/8
Conan: 2/8
Rambo: 6/6
Imie: Conan Zycie: 22
Atak fizyczny: 5 Atak magiczny: 14
Obrona fizyczna: 6 Obrona magiczna: 5

Imie: Rambo Zycie: -2
Atak fizyczny: 8 Atak magiczny: 12
Obrona fizyczna: 5 Obrona magiczna: 5
```

# Przekazywanie obiektów do funkcji - przykład

Projekt będzie rozwijany



Gdy poznamy mechanizmy takie jak:

- Dziedziczenie,
- Polimorfizm,
- Klasy abstrakcyjne

Rozbudujemy grę, tak, aby walczyły dwie drużyny składające się z postaci różnych klas (wojownik, mag, łotrzyk).



## 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