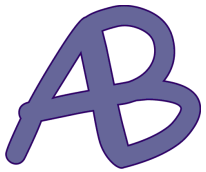


dr Artur Bartoszewski
Katedra Informatyki
UTH Radom

Pola statyczne klas

Składniki statyczne klas

Pola statyczne – stosujemy, gdy wszystkie egzemplarze obiektów danej klasy powinny współdzielić tę samą daną.

Pole statyczne jest w pamięci tworzone jednokrotnie i jest wspólne dla wszystkich egzemplarzy obiektów danej klasy.

Co więcej: istnieje nawet wtedy, gdy jeszcze nie zdefiniowaliśmy ani jednego egzemplarza obiektu tej klasy.

```
class klasa
{
  public :
    int x ;
    static int skladnik ;
};
```

Składniki statyczne klas

- ✓ Deklaracja pola statycznego w ciele klasy nie jest jego definicją.
- ✓ Definicję musimy umieścić w takim miejscu programu, aby miała zakres pliku. Czyli tak, jak definicję zmiennej globalnej.
- ✓ Definicja taka może zawierać inicjalizację.

```
int klasa::skladnik = 6;
```

- ✓ Pole statyczne może być także typu ***private***.
- ✓ Inicjalizacja pola statycznego możliwa jest nawet jeśli jest ono typu ***private***. Po inicjalizacji prywatne pole statyczne nie może być czytane ani zapisywane z poza klasy.

Składniki statyczne klas

Do składnika statycznego można odwołać się na trzy sposoby:

1. Za pomocą nazwy klasy i operatora zakresu „ :: ”

```
klasa::składnik
```

2. Jeśli istnieją już jakieś egzemplarze obiektów klasy, to możemy posłużyć się operatorem „ . ”

```
obiekt.składnik
```

3. Jeśli mamy wskaźnik do obiektu stosujemy operator „->”

```
*wsk = &obiekt;
```

```
wsk->składnik;
```

Składniki statyczne klas

```
#include <iostream>
class Klasa
{
    static int wspolne;
public:
    void metoda(){
        std::cout << wspolne << std::endl;
        ++wspolne;
    }
};

int Klasa::wspolne = 0;

int main()
{
    Klasa a, b, c;
}
```

Składniki statyczne klas

```
class osoba
{
private:
    int id;
    string imie;

public:
    static int ile_obiektow;
    osoba(string s) : imie(s)
    {
        id = ++ile_obiektow;
    }
    void setImie(string imie) { this->imie = imie; }
    string getImie() { return imie; }
    int getId() { return id; }
    static int getIleObiektow() { return ile_obiektow; }
};
int osoba::ile_obiektow = 0;

osoba *tab[DLUGOSC_TABLICY];
```

Przykład:

Klasa automatycznie nadająca kolejne numery ID nowo tworzonym obiektom

Składniki statyczne klas

Przykład:

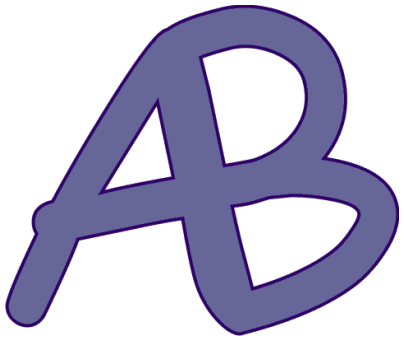
Klasa całkowicie statyczna – posiada wyłącznie składniki statyczne. Można używać jej metod bez konieczności tworzenia obiektu klasy.

```
class Stale
{
private:
    static double PI;
    static double e;

public:
    static double getPI() { return PI; }
    static double getE() { return e; }
    static double poleKola(double r) { return PI * r * r; }
};

double Stale::PI = 3.1415;
double Stale::e = 2.7182;

int main()
{
    // cout << Stale::getPI();
    int r = 10;
    double pole = Stale::poleKola(r);
    cout << pole;
    return 0;
}
```

Metoda statyczna może korzystać tylko z pól statycznych klasy.

Konieczne jest inicjalizowanie pól statecznych.

Nie ma konieczności inicjalizowania statycznych metod.



dr Artur Bartoszewski
Katedra Informatyki
UTH Radom

Funkcje zaprzyjaźnione

Funkcje zaprzyjaźnione

Funkcja zaprzyjaźniona – to funkcja która ma prawo dostępu do prywatnych składników klasy.

Funkcja zaprzyjaźniona z klasą nie jest metodą tej klasy.

- ✓ Wewnątrz definicji klasy wystarczy umieścić deklarację tej funkcji poprzedzoną słowem friend.
- ✓ Uwaga: to nie funkcja ma twierdzić, że jest zaprzyjaźniona. To klasa ma zadeklarować, że przyjaźni się z tą funkcją i nadaje jej prawo dostępu do składników prywatnych. Zatem słowo ***friend*** pojawia się tylko wewnątrz definicji klasy.
- ✓ Funkcja może być zaprzyjaźniona z wieloma klasami.

Funkcje zaprzyjaźnione

Funkcja „***funkcja()***” jest zaprzyjaźniona z klasą „***A***”

```
class A
{
    friend void funkcja(A &temp);
};

void funkcja(A &temp)
{

}
```

Aby funkcja miała możliwość modyfikacji pól klasy obiekt, dla którego zostanie wywołana powinien być przekazany przez referencję „&”.
Inaczej przekazany zostanie on przez wartość, co oznacza, że będzie funkcja będzie pracowała na kopii obiektu. Jest o możliwe, ale stwarza pewne komplikacje.

- Funkcja ma realnie tylko możliwość odczytu wartości pól.
- Podczas usuwania kopii obiektu wykonany zostanie jego destruktor.

Funkcje zaprzyjaźnione

```
class B;  //nagłówek klasy B

class A
{
    friend void f1(A &temp, B & temp2);
};

class B
{
    friend void f1(A &temp, B & temp2);
};

void f1(A &temp, B & temp2)
{

}
```

Funkcja „***f1***” jest zaprzyjaźniona z klasami „***A***” i „***B***”

Funkcje zaprzyjaźnione

```
class RGB
{
private:
    int R, G, B;
public:
    RGB(int r=0, int g=0, int b=0): R(r), G(g), B(b) {}
    friend void wypisz(RGB &k1);
};

void wypisz(RGB &k1)
{
        cout<< "R="<<k1.R<<endl
            << "G="<<k1.G<<endl
            << "B="<<k1.B<<endl;
}
int main()
{
    RGB kolor1(100,200,300);
    wypisz(kolor1);
    return 0;
}
```

Deklaracja przyjaźni z funkcją wypisz

Funkcja zaprzyjaźniona z klasą.

W parametrze przekazany jest obiekt na którym pracuje funkcja (w tym przypadku referencja do obiektu),
Funkcja ma dostęp do prywatnych pól obiektu.

Funkcje zaprzyjaźnione

Prezentuje funkcje zaprzyjaźnione pracujące na kopiach obiektów.

```
class liczba
{
private:
    int dana;

public:
    liczba(int x = 0) : dana(x)
    {
        cout << "zadzialal konstruktor\n";
    }
    int getDana() { return dana; }
    void setDana(int dana) { this->dana = dana;
}

    friend void wypisz(liczba x);
    friend bool porownaj(liczba x, liczba y);
    ~liczba()
    {
        cout << "zadzialal destruktor\n";
    }
};
```

```
void wypisz(liczba x)
{
    cout << "przekazana liczba: "
         << x.dana
         << endl;
}

bool porownaj(liczba x, liczba y)
{
    if (x.dana == y.dana)
        return true;
    else
        return false;
}
```

Funkcje zaprzyjaźnione

Prezentuje funkcje zaprzyjaźnione pracujące na kopiach obiektów C.D.

```
int main()
{
    liczba l1(100), l2(200);
    wypisz(l1);
    if (porownaj(l1, l2))
        cout << "wieksza jest liczba pierwsza\n";
    else
        cout
            << "wieksza jest liczba druga\n";
    return 0;
}
```

```
zadzialal konstruktor
zadzialal konstruktor
przekazana liczba: 100
zadzialal destruktor
zadzialal destruktor
zadzialal destruktor
wieksza jest liczba druga
zadzialal destruktor
zadzialal destruktor
```

Zauważmy, że po uruchomieniu powyższego programu konstruktor zadziałał dwukrotnie, a destruktor aż 5 razy.

Wynika to stąd, że do funkcji zaprzyjaźnionych przekazywana była kopia już istniejącego obiektu, a więc jego konstruktor nie został uruchomiony.
Destruktor obiektu uruchamiany był za każdym razem gdy usuwana była z pamięci kopia, czyli po zakończeniu działania funkcji oraz przy zakończeniu programu.

Funkcje zaprzyjaźnione

```
#include <iostream>
#include <cstring>

using namespace std;

class RGB
{
private:
    int R;
    int G;
    int B;
public:
    RGB(int r=0, int g=0, int b=0): R(r), G(g), B(b) {}
    ~RGB() {cout<<"Obiekt usuniety\n";}
    void przedstawSie();
    friend int jasnosc(RGB  &k1);
    friend RGB * suma(RGB &k1, RGB &k2);
};
```

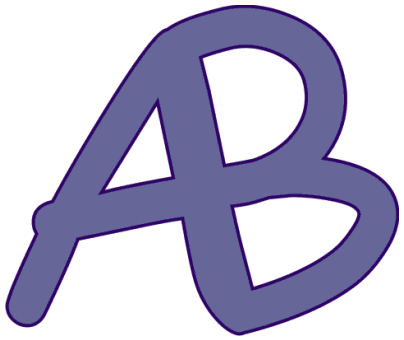
Przykład

Z klasą RGB zaprzyjaźnione są 2 funkcje

- Funkcja *jasność* oblicza średnią ze składowych kolorów.
- Funkcja *suma* oblicza sumę dwóch kolorów. Zwraca ona obiekt klasy RGB

Funkcje zaprzyjaźnione

```
void RGB::przedstawSie()
{
    cout<< "R="<<R<<endl << "G="<<G<<endl << "B="<<B<<endl;
}

int jasnosc(RGB &k1)
{
    return (k1.R+k1.G+k1.B)/3;
}

RGB * suma(RGB &k1, RGB &k2)
{
    int r =(k1.R+k2.R)/2;
    int g =(k1.G+k2.G)/2;
    int b =(k1.B+k2.B)/2;
    RGB * wynik = new RGB(r,g,b);
    return wynik;
}

int main()
{
    RGB kolor1(100,200,300), kolor2(10,20,30);
    RGB *kolorWynikowy = suma(kolor1,kolor2);
    kolorWynikowy->przedstawSie();
    delete kolorWynikowy;
    return 0;
}
```



dr Artur Bartoszewski
Katedra Informatyki
UTH Radom

Klasy zaprzyjaźnione

Klasy zaprzyjaźnione

Można zadeklarować w klasie przyjaźń ze wszystkimi metodami i polami innej klasy.

```
class B;

class A
{
    friend class B;
};
```

Metody klasy B będą miały dostęp do wszystkich składowych klasy A, ale nie odwrotnie.

Klasy zaprzyjaźnione

Wzajemna przyjaźń klas

Aby funkcje klasy A miały dostęp do składowych klasy B, a funkcje klasy B miały dostęp do składowych klasy A, możemy zaprzyjaźnić ze sobą obie klasy

```
class A;

class B
{
  friend class A;
};

class A
{
  friend class B;
};
```

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszl 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:

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