

# Algorytmy i struktury danych

Wprowadzenie do języka C++

Aleksander Lamża  
ZKSB · Instytut Informatyki  
Uniwersytet Śląski w Katowicach

[aleksander.lamza@us.edu.pl](mailto:aleksander.lamza@us.edu.pl)

- Jeszcze raz struktury
- Funkcje działające na strukturach
- Funkcje lądują w strukturach
- Obiekty!

# Jeszcze raz struktury

Założmy, że tworzymy aplikację pomagającą zarządzać magazynem w internetowym sklepie rowerowym. Musimy jakoś przechowywać informację o towarach. Tworzymy więc taką strukturę:

nazwa	Endura Singletrack II
cena_jedn	329.00
stan_mag	15

```
typedef struct {  
    char nazwa[50];  
  
    float cena_jedn;  
  
    int stan_mag;  
  
} Towar;
```

Zastanówmy się, co – poza danymi zapisanymi w strukturze – może nam się przydać w aplikacji...

# Funkcje działające na strukturach

Na pewno będziemy chcieli **wyświetlić informacje o towarze**.  
Będzie więc potrzebna funkcja `wyświetl()`.

Na pewnym etapie przyda się możliwość **obliczenia wartości towarów**.  
Do tego posłuży funkcja `obliczWartosc()`.

W każdym szanującym się sklepie są czasem **promocje i wyprzedaże**.  
Przyda się funkcja `obniżCene()` obniżająca cenę jednostkową.

# Funkcje działające na strukturach

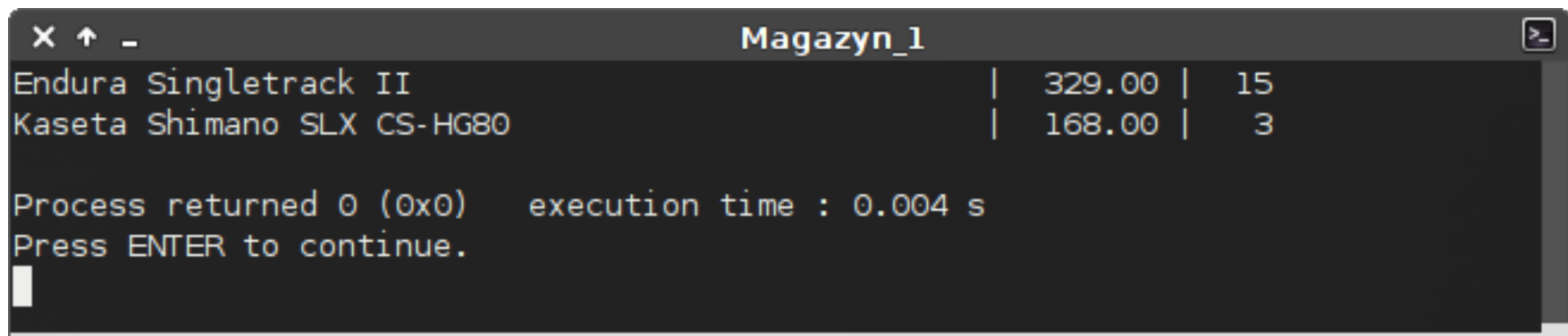
Zaczynamy od funkcji `wyswietl()`:

```
void wyswietl(Towar* towar) {  
    printf("%-30s | %7.2f | %3d\n",  
        towar->nazwa,  
        towar->cena_jedn,  
        towar->stan_mag);  
}
```

```
typedef struct {  
    char nazwa[50];  
    float cena_jedn;  
    int stan_mag;  
} Towar;
```

Czas na test:

```
int main()  
{  
    Towar t1 = {"Endura Singletrack II", 329.00, 15};  
    Towar t2 = {"Kaseta Shimano SLX CS-HG80", 168.00, 3};  
  
    wyswietl(&t1);  
    wyswietl(&t2);  
  
    return 0;  
}
```



```
Magazyn_1  
Endura Singletrack II | 329.00 | 15  
Kaseta Shimano SLX CS-HG80 | 168.00 | 3  
  
Process returned 0 (0x0) execution time : 0.004 s  
Press ENTER to continue.  
█
```

# Funkcje działające na strukturach

Funkcji `wartosc()` chciałbym używać tak:

```
int main()
{
    Towar t1 = {"Endura Singletrack II", 329.00, 15};
    Towar t2 = {"Kaseta Shimano SLX CS-HG80", 168.00, 3};

    wyswietl(&t1);
    printf("Wartość: %.2f", obliczWartosc(&t1));

    wyswietl(&t2);
    printf("Wartość: %.2f", obliczWartosc(&t2));

    return 0;
}
```

Efekt ma być taki:

```
Magazyn_1
Endura Singletrack II          | 329.00 | 15
Wartość: 4935.00
Kaseta Shimano SLX CS-HG80    | 168.00 | 3
Wartość: 504.00

Process returned 0 (0x0)   execution time : 0.007 s
Press ENTER to continue.
```

Jak powinna wyglądać funkcja `wartosc()`?

# Funkcje działające na strukturach

Oto ona:

```
float obliczWartosc(Towar* towar) {  
    return towar->cena_jedn * towar->stan_mag;  
}
```

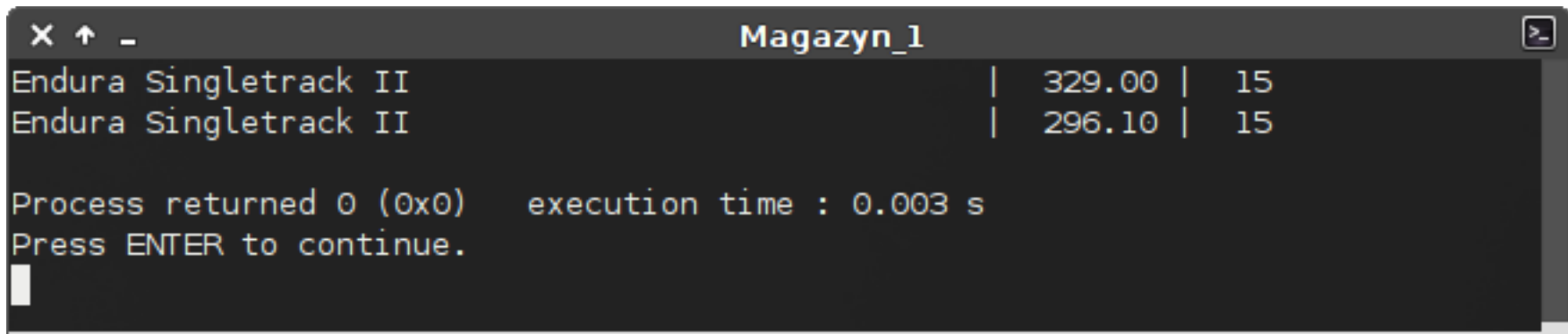
# Funkcje działające na strukturach

Została jeszcze funkcja `obnizCene()`:

```
void obnizCene(Towar* towar, int obnizka) {  
    towar->cena_jedn = towar->cena_jedn * (1.0 - obnizka/100.0);  
}
```

Testujemy:

```
int main()  
{  
    Towar t1 = {"Endura Singletrack II", 329.00, 15};  
  
    wyswietl(&t1);  
  
    obnizCene(&t1, 10);  
    wyswietl(&t1);  
  
    return 0;  
}
```



```
Magazyn_1  
Endura Singletrack II | 329.00 | 15  
Endura Singletrack II | 296.10 | 15  
  
Process returned 0 (0x0)   execution time : 0.003 s  
Press ENTER to continue.  
█
```



# Funkcje działające na strukturach

Spójrzmy teraz na przykładowy kod funkcji `main()`:

```
int main()
{
    Towar t1 = {"Endura Singletrack II", 329.00, 15};
    Towar t2 = {"Kaseta Shimano SLX CS-HG80", 168.00, 3};

    wyswietl(&t1);
    printf("Wartość: %.2f\n", obliczWartosc(&t1));

    wyswietl(&t2);
    printf("Wartość: %.2f\n", obliczWartosc(&t2));

    obnizCene(&t1, 10);
    wyswietl(&t1);

    return 0;
}
```

Na żółto wyróżniłem wszystkie wywołania funkcji.

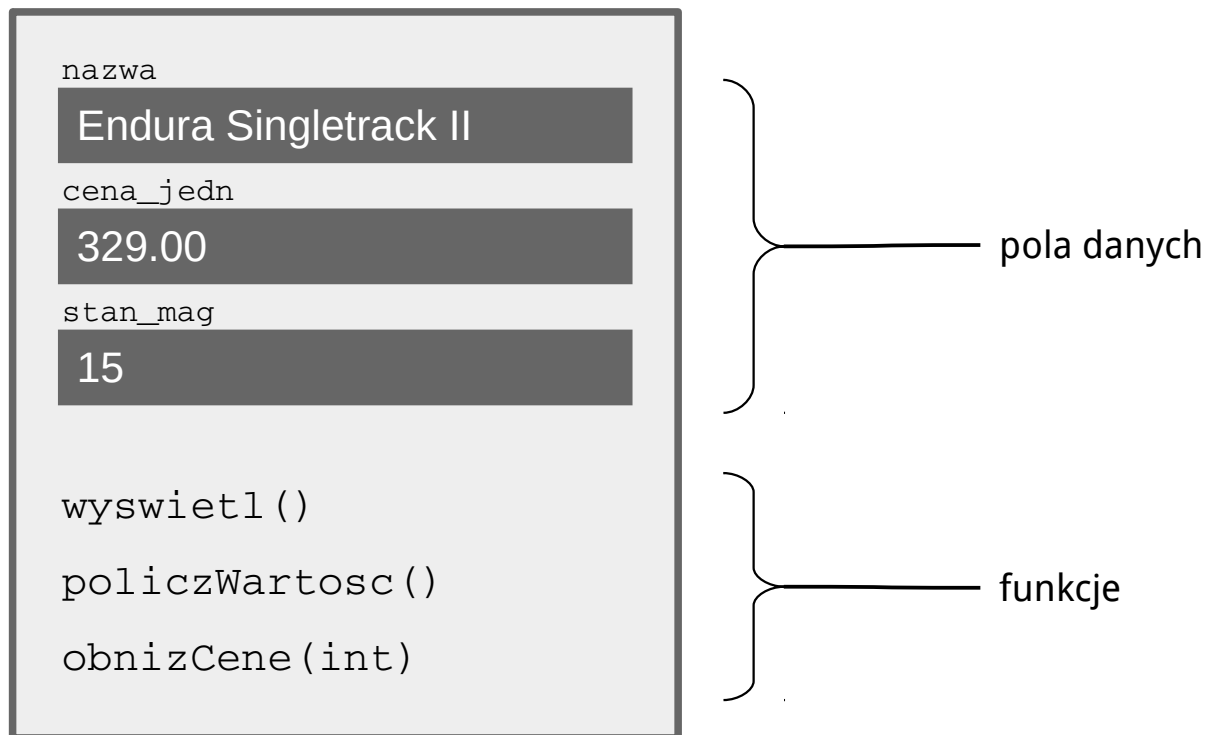
## Co się rzuca w oczy?

Każdej funkcji musimy przekazać wskaźnik na strukturę, na której ma pracować.  
To niezbyt wygodne...

## A czy jest to konieczne?

# Inne spojrzenie na funkcje i struktury

A gdybyśmy tak funkcje umieścili w strukturze?




W takim przypadku przekazywanie wskaźnika na strukturę nie jest potrzebne, ponieważ funkcja znajduje się w tej strukturze, więc wie na jakich danych ma pracować.

Zobaczmy teraz, jak to będzie wyglądało w kodzie...

# Funkcje w strukturach

```
typedef struct {  
    char nazwa[50];  
    float cena_jedn;  
    int stan_mag;  
  
    void wyswietl() { ... }  
  
    float obliczWartosc() { ... }  
  
    void obnizCene(int obnizka) { ... }  
} Towar;
```



Na razie pominąłem  
ciała funkcji.

A jak w takim razie wywołać te funkcje?

Nic prostszego:

```
Towar t1 = {"Endura Singletrack II", 329.00, 15};  
t1.wyswietl();
```

# Funkcje w strukturach

Nie wystarczy jednak tylko przenieść funkcji w niezmienionej postaci do struktury. Trzeba będzie co nieco zmienić.

Wcześniej było tak:

```
void wyswietl(Towar* towar) {  
    printf("%-50s | %7.2f | %3d\n",  
        towar->nazwa,  
        towar->cena_jedn,  
        towar->stan_mag);  
}
```

Usuwamy parametr `towar` i wszystkie jego wystąpienia:

```
void wyswietl(Towar*towar) {  
    printf("%-50s | %7.2f | %3d\n",  
        towar->nazwa,  
        towar->cena_jedn,  
        towar->stan_mag);  
}
```

I działa! Przecież wiadomo, o jakie pola `nazwa`, `cena_jedn` i `stan_mag` chodzi – te ze struktury, na której została wywołana funkcja.

# Funkcje w strukturach

To samo robimy z pozostałymi funkcjami:

```
typedef struct {  
    ... //pola  
  
    void wyswietl() {  
        printf("%-50s | %7.2f | %3d\n",  
            nazwa,  
            cena_jedn,  
            stan_mag);  
    }  
  
    float obliczWartosc() {  
        return cena_jedn * stan_mag;  
    }  
  
    void obnizCene(int obnizka) {  
        cena_jedn = cena_jedn * (1.0 - obnizka/100.0);  
    }  
} Towar;
```

A jak będzie wyglądał `main()`?

# Funkcje w strukturach

```
int main()
{
    Towar t1 = {"Endura Singletrack II", 329.00, 15};
    Towar t2 = {"Kaseta Shimano SLX CS-HG80", 168.00, 3};

    t1.wyswietl();
    printf("Wartość: %.2f\n", t1.obliczWartosc());

    t2.wyswietl();
    printf("Wartość: %.2f\n", t2.obliczWartosc());

    t1.obnizCene(10);
    t1.wyswietl();

    return 0;
}
```

Zmienne t1 i t2 możemy nazwać **obiektami**.

wyswietl(), obliczWartosc() i obnizCene() to funkcje składowe nazywane też **metodami**.

Może się wydawać, że wprowadziliśmy tylko kosmetyczne zmiany.

**To jednak dopiero początek!**

Obiektowe podejście do programowania i obiektowe mechanizmy otwierają mnóstwo wspaniałych możliwości. Dzięki nim kod jest bardziej czytelny i logiczny, a więc łatwiejszy w utrzymaniu.

Możliwości te poznacie na kolejnych wykładach,  
kiedy będziemy omawiać złożone struktury danych.