

# Programowanie komputerowe

Zajęcia 9

# Przykład – liczby wymierne

```
class Wymierna {
public:
    Wymierna() { l=0; m=1; }
    Wymierna(int n) { l=n; m=1; }
    Wymierna(int a, int b) { l=a; m=b; }
    void drukuj() { cout << l << "/" << m; }
    Wymierna suma(Wymierna w) {
        return Wymierna(l*w.m+w.l*m, m*w.m);
    }
private:
    int l,m;
};
```

# Automatyczna konwersja

- Jeśli podamy wartość, która nie ma odpowiedniego typu, kompilator próbuje ją zamienić na poprawną, np.
  - `double x=2; // 2 jest typu int, zostanie zamienione na 2.0 typu double`
  - `int n; if(n) { ... }; // n zostanie zamienione na false jeśli n=0 i true jeśli n!=0`
- Można zdefiniować konwersję z dowolnego typu  $T$  na typ klasowy  $K$ , który definiujemy. W tym celu należy zdefiniować konstruktor w klasie  $K$ , którego parametrem jest  $T$ .
- Przykład: w klasie `Wymierna` zdefiniowany jest konstruktor z parametrem `int`. Poprawne są więc wyrażenia

```
Wymierna w=5; // w będzie równa 5/1
w.suma(3); // w powiększy się o 3
```

# Przeciążanie operatorów

Żeby dodać dwie liczby wymierne  $w$  i  $x$  należy napisać

```
Wymierna y=w.suma(x);
```

co nie jest zbyt wygodne (szczególnie gdy chcemy tworzyć bardziej skomplikowane wyrażenia). Można jednak zdefiniować znaczenie standardowych operatorów, takich jak  $+$ ,  $*$ ,  $++$ ,  $==$ ,  $<$ , itd. dla klas tworzonych przez użytkownika. Można to zrobić na dwa sposoby:

- W klasie  $K$  zdefiniować metodę `operator+(T t)`. Jeśli  $k$  jest typu  $K$ ,  $t$  typu  $T$ , to wyrażenie  $k+t$  zostanie zamienione na `k.operator+(t)`.
- Zdefiniować funkcję `operator+(T t, U u)`. Wtedy  $t+u$  ( $t$  typu  $T$ ,  $u$  typu  $U$ ) zostanie zamienione na `operator+(t, u)`.
- Nie wolno używać obu sposobów jednocześnie.

# Przeciążanie operatorów – przykład

Dodawanie w klasie Wymierna.

- Sposób 1: w klasie Wymierna dodać metodę

```
Wymierna operator+(Wymierna w) {  
    return Wymierna(l*w.m+w.l*m, m*w.m);  
}
```

- Sposób 2: dodać funkcję (poza klasą Wymierna)

```
Wymierna operator+(Wymierna w1, Wymierna w2) {  
    return Wymierna(w1.l*w2.m+w2.l*w1.m, w1.m*w2.m);  
}
```

# Drukowanie obiektów

Drukowanie obiektów klasy T może być zrealizowane jako funkcja

```
ostream& operator<<(ostream& f, T x)
```

Np. dla klasy Wymierna można dodać funkcję (globalną, na zewnątrz klasy)

```
ostream& operator<<(ostream& f, Wymierna w) {  
    f << w.l << "/" << w.m;  
    return f;  
}
```

Ponieważ korzysta ona z pól prywatnych, dodajemy (w klasie Wymierna) stosowne zezwolenie (*deklarację zaprzyjaźnienia*):

```
friend ostream& operator<<(ostream& f, Wymierna w);
```

# Konwersja raz jeszcze

Jeśli chcemy umożliwić konwersję z tworzonego przez nas typu klasowego K na pewien typ T, dodajemy w klasie K metodę

```
operator T();
```

Np. konwersję Wymierna => double definiujemy dodając

```
operator double() {  
    return (double)l / m;  
}
```

**Uwaga:** (double)l oznacza wymuszenie konwersji int=>double.

# Ćwiczenia

1. Zadbać o to, żeby klasa `Wymierna` operowała na skróconych ułamkach, tj. żeby wynikiem działania  $1/9+1/9$  było  $2/9$  a nie  $18/81$ .
2. Dodać do klasy `Wymierna` operatory `-`, `*`, `/`, `==`, `<`, `+=`, itd.
3. Napisać program, który pyta użytkownika o liczbę wymierną, a następnie przedstawia ją w postaci sumy różnych ułamków postaci  $1/n$ .
4. Napisać klasę `Zespolona` do obsługi liczb zespolonych.