Przeciążanie operatorów

Operatory które można przeciążyć w C#

- Operatory jednoargumentowe:
  `+, -, !, ~, ++, --, true, false`
- Operatory relacji:
  `==, !=, <, >, <=, >=`
- Pozostałe operatory dwuargumentowe:
  `+, -, *, /, %, &, |, ^, <<, >>`
- Operator konwersji:
  `()`

Operatory przeciążane pośrednio

- Złożone operatory przypisania:
  `+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- Operatory logiczne: `&&` oraz `||`
  `x || y ::= true(x) ? x : x | y`
  `x && y ::= false(x) ? x : x & y`
- Operator `[]`
  indeksator

Metoda definiująca operator: zasady ogólne

- Musi być metodą publiczną
- Musi być metodą statyczną
- Co najmniej typ jednego z argumentów lub typ wartości zwracanej musi być określony przez klasę lub strukturę w której dana metoda jest zdefiniowana
- Metoda musi coś zwracać, typ wartości przekazywanej nie może być void
- Wszystkie argumenty metody muszą być przekazywane przez wartość
- Nazwa metody musi mieć format: operator op, gdzie op jest symbolem operatora

Operatory jednoargumentowe: +, −, !, ~

- Typ jedynego argument przesyłanego do metody definiującej dany operator, musi być określony przez klasę lub strukturę w której dana metoda jest zdefiniowana

- Typem zwracanym może być dowolny typ (poza void)

```
struct MojaLiczba
{
   public int Liczba;
   public static double operator ~(MojaLiczba x)
   {
     return 1.0 / x.Liczba;
   }
}
```

Operatory jednoargumentowe: ++, --

- Typ jedynego argumentu i typ wartości przekazywanej musi być określony przez klasę lub strukturę w której dana metoda jest zdefiniowana

- Pojedyncza metoda definiuje zarówno w wersję przyrostkową jak i przedrostkową

```
struct MojaLiczba
{
   public int Liczba;
   public static MojaLiczba operator ++(
                                  MojaLiczba x)
   {
      x.Liczba++;
      return x;
   }
}
```

Operatory jednoargumentowe: true, false

- Typ jedynego argument przesyłanego do metody definiującej dany operator, musi być określony przez klasę lub strukturę w której dana metoda jest zdefiniowana

- Typem zwracanym musi być **bool**

- Muszą być przeciążane razem

```
struct MojaLiczba {
   public int Liczba;
   public static bool operator true(MojaLiczba x) {
      return x.Liczba != 0;
   }
   public static bool operator false(MojaLiczba x) {
      return x.Liczba == 0;
   }
}
```

Operatory relacji

- Typem zwracanym nie musi być **bool**

- Muszą być przeciążane parami: == i !=, < i >, <= i >=

```
struct MojaLiczba {
   public int Liczba;
   public static bool operator <(MojaLiczbax, int y){
      return x.Liczba < y;
   }
   public static bool operator >(MojaLiczba x, int y){
      return x.Liczba > y;
   }
}
```

Operatory binarne

- Operatory << i >>
  - pierwszy argument musi być typu określonego przez klasę lub strukturę, dla której definiujemy dany operator
  - drugi argument musi być typu **int**

```
struct MojaLiczba {
   public int Liczba;
   public static MojaLiczba operator <<
       (MojaLiczba x, int y)
   {
      MojaLiczba z ;
      z.Liczba = x.Liczba + y;
      return z;
   }
}
```

Operatory konwersji

```
public static <typKonwersji> operator typ1 (
                                    typ2 nazwaArgumentu)
{ ...
}
```

- typ_konwersji
  - **implicit**: konwersja niejawna
  - **explicit**: konwersja jawna
- Jeden z typów typ1 albo typ2 musi być określony przez klasę lub strukturę, w której dana metoda jest zdefiniowana