Zadanie 5

Urządzany jest turniej. Zawodnicy współpracujący w kilkuosobowych grupach mają łącznie przejechać tysiąc kilometrów na monocyklach. Codziennie podsumowywana jest suma zrobionych przez zawodników kilometrów. Tworzymy program do łatwiejszego zarządzania całością:

1. Napisać **klasę `Zawodnik`** (trzy prywatne pola składowe: imię, ilość dni w danym momencie, oraz tablicę przechowującą informację ile każdego dnia zostało zrobionych kilometrów) + konstruktor główny (z trzema parametrami), konstruktor z jednym parametrem (imieniem, w takim wypadku 1 dzień) oraz konstruktor domyślny (ustawiamy pola na wymyślone przez siebie wartości). Destruktor i konstruktor kopiujący jeśli potrzebny.
2. Napisać funkcję składową **`DodajDzien(double km)`** - zwiększającą ilość dni o jeden oraz wypełniającą odpowiednią pozycję w tablicy wartością km.
3. Przeładować **operator dodawania `operator+`** wewnątrz klasy, tak by zwracał sumę wszystkich km przejechanych przez obydwu zawodników (z jednym argumentem)
4. Przeładować **operator przypisania `operator=`** (zgodnie ze zdrowym rozsądkiem)
5. Przeładować **`operator[]`** w ten sposób, by pozwalał na przypisanie oraz zwrócenie liczby km przejechanych danego dnia, podanego jako argument.
6. Napisać funkcję składową **`Srednia()`** - zwracającą średnią liczbę km na dzień (biorąc pod uwagę wszystkie dni) dla danego zawodnika.
7. Napisać funkcje składowe **„set”** i **„get”** dla składnika „imię”.
8. Przeładować **`operator<<`** w celu wypisania informacji o zawodniku.

Funkcja **main** powinna wyglądać następująco:

```
int main()
{
  Zawodnik z1("Jan");
  cout<<z1;

  double km[] = {40,32,43,12,54};
  Zawodnik z2("Adam",km,5);
  cout<<z2;

  Zawodnik z3(z2);    //1 pkt
  z3.setImie("Ewa");
  z3.DodajDzien(66);
  cout<<z3;        //2 pkt

  cout<<"Suma km:"<<z2+z3<<endl; // 3 pkt

  z1 = z2;     //4 pkt
  z1.setImie("Jan");
  z1[0] = 0;
  cout<<z1; //5 pkt
  cout<<z1.Srednia()<<endl;

  return 0;
}
```

Operatory +, -, *, /
Mogą być przeładowane w wersji jedno- [wewnątrz klasy] lub dwuargumentowej [na zewnątrz klasy]. Tak jak w przypadku każdej innej funkcji nie istnieją ograniczenia co do zwracanych wartości: operator taki może zwracać obiekt tej samej klasy, którą dodaje (np. możemy napisać klasę która dodając do siebie dwa obiekty klasy człowiek zwróci również człowieka), lub dowolny inny, jaki sobie wymyślimy (dodając do siebie dwóch ludzi zwróci nam np. sumę ich wieku).

Operator przypisania = (za wykładem)
Dwuargumentowy operator przypisania = służy do przypisanie jednemu obiektowy klasy treści drugiego obiektu tej samej klasy.

```
klasa & klasa::operator=(klasa &);
```

Jeśli operator= nie zostanie zdefiniowany, to zostanie on automatycznie wygenerowany (**podobnie jak konstruktor kopiujący**!), przepisane zostaną pola "składnik po składniku" (podobnie jak w przypadku automatycznie generowanego konstruktora kopiującego). W rezultacie takiego przypisania będą dwa obiekty o bliźniaczej treści.
Kłopoty natomiast mogą pojawić się wtedy, kiedy **składnikami klasy są wskaźniki** lub jeśli klasa używa **operatora new do rezerwacji miejsca w zapasie pamięci**.
Operator przypisania składa się z części:
(1) sprawdzenie, czy nie jest kopiowane siebie samego
(2) części "destruktorowej" (np. zwalnianie pamięci)
(3) części "konstruktorowej", przypominającej konstruktor kopiujący
Przykład użycia: **(jeśli operator znajduje się wewnątrz klasy):**

```
  Nazwa & operator=(Nazwa & na){;
    if (&na == this) return *this; //sprawdzenie, czy nie kopiuje samego siebie
    delete X; //zwalniamy pamięć dla rzeczy, które mogły mieć poprzednio zaalokowaną pamięć
    //alokujemy nową pamięć i przepisujemy wartości tak samo jak w konstruktorze kopiującym
    return *this;
 }
```

`operator [ ]`
Operator [ ] to operator odwołania się do elementu tablicy, jest dwuargumentowy. Chcąc napisać funkcję operatorową [ ], tak, aby operator [ ] mógł stać po obu stronach znaku = musimy zadeklarować, że funkcja ta będzie zwracała referencję do tego, czemu mamy przypisywać!
Przykład implementacji operatora [ ]:

```
int operator[](int i) { return a[i]; }
```

`operator<<`
Operator << to operator strumienia wyjścia, który możemy wykorzystać np. w komendzie cout służącej do wypisania na ekran informacji (domyślnie tylko predefiniowane typu zostaną wypisane). Operator ten musi być przeciążony jako operator zewnętrzny (będący funkcją globalną). Przykład implementacji:

```
ostream& operator<<(ostream& wyjscie, const Klasa& k)
{
      wyjscie<<"Liczba: "<<k.liczba<<endl;
      return wyjscie;
}
```

Dodatkowe:
Napisać opearator odejmowania (`operator-`) na zewnątrz klasy, jako funkcję globalną (przyjmujący dwa argumenty). Powinien zwracać różnicę w przejechanych kilometrach.
Napisać `operator++`, który zwiąksza ilość kilometrów przejechanych ostatniego dnia o 10.