Zadanie 6, Języki Programowania, 21.11.2013

Urządzany jest wyścig. Zawodnicy współpracujący w kilkuosobowych grupach mają łącznie przejechać 100 tysięcy kilometrów. Codziennie podsumowywana jest suma zrobionych przez zawodników kilometrów. Tworzymy program do łatwiejszego zarządzania całością:

1. Napisać **klasę Pojazd** (trzy prywatne pola składowe: imię właściciela, ilość dni oraz tablicę przechowującą informację ile każdego dnia zostało zrobionych kilometrów) + konstruktor główny (z trzema parametrami) + konstruktor z jednym parametrem (imieniem, w takim wypadku 0 dni). Destruktor i konstruktor kopiujący jeśli potrzebne.
2. Napisać funkcje składowe **Wypisz** (wypisujące informacje o Pojeździe na ekran) oraz „**set**" dla składnika „imię".
3. Przeładować **operator dodawania +** wewnątrz klasy, tak by zwracał sumę wszystkich km przejechanych przez obydwu zawodników (zaimplementować wersję z jednym argumentem)
4. Przeładować **operator przypisania =** (zgodnie ze zdrowym rozsądkiem)
5. Przeładować **operator [ ]** w ten sposób, by pozwalał na *przypisanie oraz zwrócenie* liczby km przejechanych danego, podanego dnia.
6. Napisać funkcję składową **Srednia()** - zwracającą średnią liczbę km na dzień (biorąc pod uwagę wszystkie dni) dla danego zawodnika.
7. Napisać funkcję składową **DodajDzien(double km)** - zwiększającą ilość dni o jeden oraz wypełniającą odpowiednią pozycję w tablicy wartością km.

Operatory +, -, *, /

Mogą być przeładowane w wersji jedno- [wewnątrz klasy] lub dwuargumentowej [na zewnątrz klasy]. Nie istnieją ograniczenia co do typu 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 użycia:

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

3. Należy w funkcji main po kolei:

```
int main()
{

  Pojazd p1("Kacper");
  p1.Wypisz();

  double km[] = {4320,3432,4293,1002,5114};
  Pojazd p2("Krzysiek",km,5);
  p2.Wypisz();

  Pojazd z3(z2);
  p3.setImie("Marta");
  cout<<p3.Srednia()<<endl;
  p3.Wypisz();

  cout<<"Suma dni:"<<p2+p3<<endl;

  p1 = p2;
  p1.setImie("Jan");
  p1[0] = 0;
  p1.Wypisz();

  p3.DodajDzien(66);
  p3.Wypisz();
  return 0;
}
```

Zawsze należy owijać definicje klas w plikach nagłówkowych w strukturę „ifndef”:

```
#ifndef _NAZWA_TOKENU
#define _NAZWA_ TOKENU
       class klasa{...}; - definicja naszej klasy
#endif
```

Działa to w ten sposób: jeśli token _NAZWA_TOKENU nie był jeszcze definiowany: wejdź do środka (zdefiniuj token i zapisz go sobie w pamięci, po czym wykonaj wszystkie instrukcje znajdujące się w środku). Jeśli już go deklarowałeś – przejdź od razu do #endif. W ten sposób już nigdy nie będziemy się przejmować kolejnością ładowanych bibliotek – jeśli zostały one wcześniej załadowane, kompilator w ogóle pominie taką instrukcje.

Należy skompilować tak przygotowany program używając w linii poleceń komendy: `make`.

```
CXX = g++
CXXFLAGS = -Wall
LFLAGS =

OBJS = Zawodnik.o main.o

all: prog

prog: $(OBJS)
       $(CXX)  $(CXXFLAGS)  $^  -o  $@

clean:
       rm  -f  *.o  prog

.PHONY:  all  clean
```

Dodatkowe:

Napisać opearator odejmowania (operator-) na zewnątrz klasy (przyjmujący dwa argumenty). Powinien zwracać różnicę w przejechanych kilometrach.