Zadanie D - Natężenie ruchu
Punktów procentowych do uzyskania: **8**
Język programowania: C++
autorka zadania: Magdalena Wiercioch

Opis zadania

- Przewidywanie natężenia ruchu pojazdów pełni ważną rolę w systemach sterowania ruchu drogowego. Większość technik predykcji (naukowego prognozowania) ruchu drogowego opiera się na wcześniej zebranej informacji i wykorzystuje się przy tym wiele algorytmów mających swoje źródło w statystyce, uczeniu maszynowym i innych dziedzinach wiedzy. Rozwiązujący zadanie jest członkiem zespołu programistycznego, pracującego nad stworzeniem systemu analizującego natężenie ruchu.
- Z informacji dostarczonych przez klienta wiadomo, że dane natężenia ruchu rejestrowane są przez detektory umieszczone w analizowanych lokalizacjach. Każdy detektor gromadzi dane w pewnych odstępach czasu, co powoduje uzyskanie zestawu (wektora) wartości natężenia ruchu, a więc ilości pojazdów przejeżdżających w danym interwale. Kluczową motywacją zastosowania czujników jest jednak dostarczenie przewidywanej wartości natężenia ruchu w kolejnym odcinku czasu korzystając z dotychczasowych obserwacji.
- Zadaniem jest opracowanie zestawu klas, które pozwolą na odtworzenie wykonywanych pomiarów ruchu i wskazanie na potrzeby zadania bardzo uproszczonej informacji do prognozy ruchu na przyszłość.

Wymagane klasy

- **class** PunktKontroli
  Klasa reprezentuje dane zebrane w określonym punkcie kontroli, czyli dostarczone przez pojedynczy detektor. Przechowuje wykonane pomiary, a dodatkowo każdy czujnik ma zadaną stałą liczbę pomiarów do wykonania. Może posiadać jedynie niepubliczne pola (atrybuty), a ponadto musi dostarczać poniższych publicznych metod o nagłówkach:
  - `PunktKontroli ( const string& nazwa, const int ilosc );`
    Konstruktor klasy `PunktKontroli` nadaje nazwę detektorowi (pierwszy argument), inicjalizuje stałe pole (`const`) odnoszące się do wskazanej przez drugi argument liczby wszystkich pomiarów do wykonania. Ponadto alokuje pamięć dla tablicy wskaźników do obiektów opisanej dalej klasy `Pomiar` (tablica wskaźników ma rozmiar `ilosc`).
  - `int zwrocIloscPomiarowDoWykonania() const;`
    Zwraca stałą dla danego detektora liczbę pomiarów do wykonania.
  - `Pomiar* const *pobierzPomiary();`
    Zwraca pomiary zgodnie z kolejnością występowania.
  - `static int iloscPunktowKontroli ();`
    Zwraca ilość utworzonych obiektów dokładnie klasy `PunktKontroli`.
  - `void zarejestrujPomiar ( double wartoscZmierzona, int rok, int miesiac, int dzien, int godzina, int minuta, int sekunda);`
    O ile liczba pomiarów dla danego detektora nie została przekroczona, to umieszcza pomiar (wskaźnik do obiektu `pomiar`) w tablicy wskaźników do pomiarów na kolejnej wolnej pozycji.
  - `void komunikat () const;`
    Wypisuje komunikat dotyczący danych zarejestrowanych w punkcie kontroli poprzez wypisanie w jednej linii nazwy punktu kontroli oraz znaku dwukropka (bez spacji pomiędzy). W kolejnych liniach powinna zostać wypisana informacja o każdym zapisanym pomiarze - zgodnie z opisem metody `informacja` klasy `Pomiar`.
  - `double predykcja ( PunktKontroli** pk, int k, int iloscPK );`
    Zwraca średnią arytmetyczną odległości do podanej w drugim argumencie ilości najbliższych punktów spośród dostarczonych w parametrze pierwszym (*k* pierwszych elementów tablicy). Trzeci argument to rozmiar tablicy podanej w pierwszym argumencie. Więcej w części opisu *Określenie predykcji*.
- **class** Czas
  Reprezentuje dokładny czas wykonania pomiaru, a więc rok, miesiąc, dzień, godzinę, minutę oraz sekundę, będąc klasą zagnieżdżoną klasy `Pomiar`. Może posiadać jedynie prywatne pola (atrybuty), a ponadto musi dostarczać poniższych publicznych metod o nagłówkach:
  - `Czas ();`
    Konstruktor domyślny - dzień, miesiąc, rok, godzina, minuta oraz sekunda wynoszą zero.
  - `Czas ( int rok, int miesiac, int dzien, int godzina, int minuta, int sekunda );`
    Konstruktor klasy `Pomiar::Czas`.
  - `void wypisz () const;`
    W jednej linii wypisuje komunikat dotyczący czasu wykonania pomiaru w formacie:
    `dd-mm-rrrr gg:mm:ss`
- **class** Pomiar
  Klasa opisuje pojedynczy pomiar posiadający pewną wartość oraz czas wykonania. Może zawierać jedynie niepubliczne pola (atrybuty) i musi dostarczać poniższych publicznych metod o nagłówkach:
  - `Pomiar ( double w, const Czas& c );`
    Konstruktor klasy `Pomiar` działający w oparciu o zmierzoną wartość natężenia ruchu oraz czas pomiaru.
  - `void informacja () const;`
    Dla danego pomiaru w pierwszej linii wypisuje czas jego wykonania, zgodnie z wymaganiami opisanymi w metodzie `wypisz` klasy `Czas`. W kolejnej linii wypisuje na standardowe wyjście wartość pomiaru.
  - `const Czas& dataPomiaru () const;`
    Zwraca pełny czas wykonania pomiaru, a więc z datą i godziną.
  - `double pobierzWartosc() const;`
    Zwraca wartość pomiaru.
- **class** PunktKontroliBeta
  Z uwagi na fakt, iż klasa `PunktKontroli` reprezentuje podstawowy typ opisujący informacje zgromadzone przez jeden detektor, możliwe jest jej rozszerzenie. Dlatego klasa `PunktKontroliBeta` dziedziczy w sposób publiczny po klasie `PunktKontroli`, jednakże posługuje się inną predykcją natężenia ruchu. W odróżnieniu od metody `predykcja` z klasy `PunktKontroli` prognoza w klasie `PunktKontroliBeta` odbywa się według metryki euklidesowej.

Określenie predykcji

Metoda `predykcja` ma na celu zwracanie średniej arytmetycznej odległości do ustalonej liczby najbliższych punktów sąsiadujących. W tym celu należy znaleźć wskazaną (poprzez trzeci parametr metody) ilość sąsiadów, a więc punktów, które są najbardziej „podobne” do aktualnego punktu kontroli (w sensie uzyskanych pomiarów). Aktualny punkt kontroli reprezentowany jest przez obiekt na rzecz którego wywołano metodę `predykcja`, natomiast podobieństwo między punktami kontroli wyznaczane jest przy pomocy metryki (uogólnionej odległości). W metodzie `predykcja` klasy `PunktKontroli` należy zastosować metrykę uliczną (metrykę Manhattan), zaś w metodzie `predykcja` klasy `PunktKontroliBeta` należy zastosować metrykę euklidesową. Wartość zwracana powinna zostać zaokrąglona w dół, dopuszczalne jest użycie funkcji `floor`, a tym samym włączenie pliku nagłówkowego `cmath`.

W przestrzeni *n*-wymiarowej dla danej pary punktów oznaczonych *x* oraz *y*:

- Metrykę uliczną określa formuła:

$$d(x,y)=\sum_{i=1}^{n}\left|x_i-y_i\right|$$

- Metrykę euklidesową określa formuła:

$$d(x,y)=\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2}\,.$$

Informacje dodatkowe

- Rozwiązanie należy przesłać w postaci archiwum **zip**. W głównym katalogu archiwum powinny znaleźć się tylko trzy pliki: `Pomiar.h`, `PunktKontroli.h` oraz `PunktKontroliBeta.h` zawierające definicje odpowiednich klas. O ile istnieje taka potrzeba, oczekiwane jest dostarczenie destruktorów i być może przydatne okaże się użycie słów kluczowych **`friend`** oraz **`virtual`**.
- Jedyne dozwolone do włączenia pliki nagłówkowe to: `iostream` oraz `cmath` (tam, gdzie to konieczne). Każde inne użycie wyrażenia w formie `#include` (w komentarzach lub gdziekolwiek indziej w kodzie) będzie sygnalizowane błędem kompilacji.

Test jawny

Przykładowy kod źródłowy

```
#include <iostream>
#include "Pomiar.h"
#include "PunktKontroli.h"
#include "PunktKontroliBeta.h"

using namespace std;

int main () {
    int ile_sasiadow = 2;
    string nazwy_punktow [] = { "A", "B", "C", "D", "E", "F", "G" };
    int ile_punktow_pomiaru = 5;
    PunktKontroli** punktyKontroli = new PunktKontroli* [ ile_punktow_pomiaru ];

    punktyKontroli [ 0 ] = new PunktKontroli ( "A", 4 );
    punktyKontroli [ 0 ]->zarejestrujPomiar ( 25, 2015, 11, 9, 13, 9, 6 );
    punktyKontroli [ 0 ]->zarejestrujPomiar ( 20, 2015, 11, 9, 13, 39, 6 );
    punktyKontroli [ 0 ]->zarejestrujPomiar ( 17, 2015, 11, 9, 14, 9, 7 );
    punktyKontroli [ 0 ]->zarejestrujPomiar ( 30, 2015, 11, 9, 14, 39, 6 );

    punktyKontroli [ 1 ] = new PunktKontroli ( "B", 4 );
    punktyKontroli [ 1 ]->zarejestrujPomiar ( 21, 2015, 11, 12, 13, 9, 6 );
    punktyKontroli [ 1 ]->zarejestrujPomiar ( 19, 2015, 11, 12, 13, 39, 6 );
    punktyKontroli [ 1 ]->zarejestrujPomiar ( 26, 2015, 11, 12, 14, 9, 7 );
    punktyKontroli [ 1 ]->zarejestrujPomiar ( 31, 2015, 11, 12, 14, 39, 6 );

    punktyKontroli [ 2 ] = new PunktKontroli ( "C", 4 );
    punktyKontroli [ 2 ]->zarejestrujPomiar ( 20, 2015, 11, 13, 13, 9, 6 );
    punktyKontroli [ 2 ]->zarejestrujPomiar ( 22, 2015, 11, 13, 13, 39, 6 );
    punktyKontroli [ 2 ]->zarejestrujPomiar ( 20, 2015, 11, 13, 14, 9, 7 );
    punktyKontroli [ 2 ]->zarejestrujPomiar ( 26, 2015, 11, 13, 14, 39, 6 );

    punktyKontroli [ 3 ] = new PunktKontroli ( "D", 4 );
    punktyKontroli [ 3 ]->zarejestrujPomiar ( 23, 2015, 11, 14, 13, 9, 6 );
    punktyKontroli [ 3 ]->zarejestrujPomiar ( 22, 2015, 11, 14, 13, 39, 6 );
    punktyKontroli [ 3 ]->zarejestrujPomiar ( 17, 2015, 11, 14, 14, 9, 7 );
    punktyKontroli [ 3 ]->zarejestrujPomiar ( 28, 2015, 11, 14, 14, 39, 6 );

    punktyKontroli [ 4 ] = new PunktKontroli ( "E", 4 );
    punktyKontroli [ 4 ]->zarejestrujPomiar ( 25, 2015, 11, 16, 13, 9, 6 );
    punktyKontroli [ 4 ]->zarejestrujPomiar ( 21, 2015, 11, 16, 13, 39, 6 );
    punktyKontroli [ 4 ]->zarejestrujPomiar ( 18, 2015, 11, 16, 14, 9, 7 );
    punktyKontroli [ 4 ]->zarejestrujPomiar ( 26, 2015, 11, 16, 14, 39, 6 );

    for ( int i = 0; i < ile_punktow_pomiaru; i++ )
        punktyKontroli [ i ]->komunikat ();

    PunktKontroli punktKontroli1 = PunktKontroli ( "F", 4 );
    punktKontroli1.zarejestrujPomiar ( 23, 2016, 1, 17, 13, 9, 6 );
    punktKontroli1.zarejestrujPomiar ( 22, 2016, 1, 18, 13, 39, 6 );
    punktKontroli1.zarejestrujPomiar ( 17, 2016, 1, 19, 14, 9, 7 );
    punktKontroli1.zarejestrujPomiar ( 28, 2016, 1, 20, 14, 39, 6 );

    punktKontroli1.komunikat ();

    cout << punktKontroli1.predykcja ( punktyKontroli, ile_sasiadow, ile_punktow_pomiaru ) << endl;

    delete [] punktyKontroli;

    return 0;
    }
```

Oczekiwane wyjście

```
A:
09-11-2015 13:09:06
25
09-11-2015 13:39:06
20
09-11-2015 14:09:07
17
09-11-2015 14:39:06
30
B:
12-11-2015 13:09:06
21
12-11-2015 13:39:06
19
12-11-2015 14:09:07
26
12-11-2015 14:39:06
31
C:
13-11-2015 13:09:06
20
13-11-2015 13:39:06
22
13-11-2015 14:09:07
20
13-11-2015 14:39:06
26
D:
14-11-2015 13:09:06
23
14-11-2015 13:39:06
22
14-11-2015 14:09:07
17
14-11-2015 14:39:06
28
E:
16-11-2015 13:09:06
25
16-11-2015 13:39:06
21
16-11-2015 14:09:07
18
16-11-2015 14:39:06
26
F:
17-01-2016 13:09:06
23
18-01-2016 13:39:06
22
19-01-2016 14:09:07
17
20-01-2016 14:39:06
28
3
```