Zadania z 3. części materiału - do poćwiczenia przed egzaminem

1. Pięciu turystów wyrusza ze wspólnego schroniska. Przyjmij dla uproszczenia, że poruszają się na płaszczyźnie (X,Y). Każdy z nich przeszedł 4 odcinki, a w każdym z odcinków przemieścił się o (ΔX, ΔY) – każda współrzędna przemieszczenia niech będzie liczbą losową w przedziale [1 .. 3] (km).

Twoim zadaniem jest policzenie sumarycznej drogi każdego turysty i wypisanie jej na ekran. Wykonaj to jednak, korzystając z kontenerów STL, a wypełnianie i liczenie drogi wykonaj, używając narzędzi biblioteki <algorithm> .

Propozycja wykonania:

- Dla jednego turysty, każde przemieszczenie (ΔX, ΔY) może być obiektem pair<float,float>, a zbiór odcinków turysty może być vector'em tych par. Pojemnikiem przechowującym dane wszystkich turystów też może być vector.
- Pomyśl, jakie narzędzie z <algorithm> dobrze wykona zadanie wypełniania kontenera. To narzędzie zaaplikuje funkcję na każdym elemencie kontenera. Pomyśl, co ta funkcja ma przyjąć, a co zwrócić.
- Tablicą dróg turystów może też być vector. Zatem działanie obliczenia drogi dla każdego z turystów jest transformacją z kontenera-źródła do kontenera-celu. Użyj tu adekwatnego narzędzia z <algorithm>, które na każdym elemencie kontenera-źródła wywoła dedykowaną funkcję. Co ta funkcja ma przyjąć, a co zwrócić? W tej funkcji danemu zawodnikowi posumuj odcinki w całą drogę.

2. Dla kolejnych współrzędnych X: 0, 1, …, 7 wylosuj współrzędne Y jako liczby zmiennoprzecinkowe z przedziału [0, 10]. Punkty te przybrały (lub nie) układ podobny do linii prostej. Jedną z miar liniowości jest współczynnik r Pearsona:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Przybiera on wartości ∈ [-1 … 0 … 1], przy czym im bardziej punkty układają się w linie prostą, tym bardziej $|r| \to 1$.

Wyobraź sobie wszystkie możliwe permutacje zestawu Y_i dla ustalonych współrzędnych X_i. Przeiteruj się przez kolejne permutacje. Każdej z nich można obliczyć współczynnik r. Znajdź taką permutację, w której układ punktów jest najbardziej liniowy ze wszystkich: podaj dla niej wartość współczynnika *r* i wypisz taki zestaw punktów. Aby zestawowi punktów policzyć współczynnik r metodą blokową, do przechowania zestawu współrzędnych X i zestawu Y użyj obiektów typu valarray. Zestaw punktów możesz połączyć w pair<valarray<..>, valarray<..>>.

Wskazówki:

- Jaka funkcja podaje liczbę permutacji zbioru N-elementowego. Dodaj ją do kodu.
- Warto utworzyć osobną funkcję, która przyjmuje dane z zestawem punktów i oblicza na ich podstawie współczynnik *r*. Działania blokowe na obiektach valarray znacznie uproszczą jej postać.
- Użyj dedykowanego narzędzia w bibliotece <algorithm> do wykonania kolejnej permutacji na dowolnej tablicy danych.

3. Utwórz strukturę Person z polami imię, nazwisko typu string. Dodaj jej operator< między obiektem rodzimym a drugim P2, który zwraca prawdę, gdy nazwisko w obiekcie rodzimym jest wcześniejsze alfabetycznie, a jeśli nazwiska są identyczne, to gdy wczesniejsze jest imię w obiekcie rodzimym.

W funkcji main() utwórz tablicę obiektów Person, przypisując jej np. zbiór takich osób: Jan Przykładny, Anna Iksinska, Bartosz Przykładny. Teraz wstaw te osoby do obiektu STL set.
Uwaga: wymogiem jest, aby operator< w strukturze Person na końcu nagłówka miał const.
Na koniec wypisz z set imiona i nazwiska osób: sprawdź, czy sortowanie zaszło prawidłowo.

4. W tym pliku znajdziesz zestawienie kilku miast i odpowiadających im temperatur miesięcznych.

- Utwórz mapę Data i wypełnij ją powyższymi danymi
  (klucz = nazwa miasta, wartość = vector jego temperatur).

- Używając narzędzia z algorithm, przeiteruj się po Data, wywołując funkcję Print_city.
  Napisz tę funkcję, aby dla każdego elementu Data wypisała: nazwę i zestaw temperatur.
  Używając narzędzia z iomanip, sformatuj czytelnie ten wypis.

- Następnie, w kodzie znajdziesz mapę „Temps". Kluczem ma tu być miasto, a wartością – tuple z trzema temperaturami dla niego: najchłodniejszą, średnią i najcieplejszą. Przeiteruj się po elementach mapy Data, a na każdym wywołaj funkcję Analyse, którą też zakoduj. Jej celem ma być wyznaczenie dla danej pary {miasto, temperatury} – powyższych trzech temperatur. Funkcja Analyse ma je zwrócić jako tuple, abyś mógł/mogła przypisać taką tuplę jako wartość dla klucza = nazwy miasta.

- Zakoduj teraz funkcję Print_Ana, która przyjmie mapę Temps i wypisze dla każdego z miast powyższe 3 temperatury. Również tu zadbaj o czytelny format wypisu.

- Operując na mapie Temps, znajdź nazwę miasta najchłodniejszego i najcieplejszego. Używajac metody {mapa}.erase ({klucz}) skasuj te miasta z mapy Temps.

5. Rozważ rzut pionowy w górę jako uogólnienie spadku swobodnego, a rzut ukośny – jako uogólnienie rzutu pionowego. W tym duchu zakoduj struct Pionowy jako strukturę dziedziczącą po struct Swobodny, a struct Ukosny – jako dziedziczącą po Pionowy. Używaj wszędzie typu float.

W najbardziej bazowej strukturze, Swobodny, zadeklaruj pola wspólne dla wszystkich struktur: g (przyspieszenie ziemskie) , Y0 (wysokość początkowa) , tz (chwila upadku na Ziemię) i Vyz (prędkość w chwili upadku na Ziemię). Zakoduj tu konstruktor, który przyjmie g i Y0 – i na tej bazie obliczy tz oraz Vyz (z = Ziemia) jako:

$$t_{Ziemia} = \sqrt{\frac{2 \cdot Y_0}{g}} \qquad V_{Y,Ziemia} = V_Y\left(t_{Ziemia}\right)$$

Teraz zakoduj wirtualne metody Y (float t) i Vy (float t), zwracające wysokość i prędkość w funkcji chwili czasu t, według wzorów:

$$Y(t) = Y_0 - \frac{g t^2}{2} \qquad V_Y(t) = -g \cdot t$$

Na koniec zakoduj wirtualną metodę Status, która czytelnie wypisze: pola deklarowane przez konstruktor (g, Y0) oraz pola wyliczane na podstawie tych pierwszych (tz, Vyz).

Przejdź do konstrukcji struct Pionowy jako dziedziczącej po Swobodny. Nowymi polami są tu: V0y (prędkość pionowa w chwili początkowej), Ymax (wysokość maksymalna) i t_Ymax (chwila, w której ciało osiągnęło Ymax) . Zaimplementuj konstruktor wypełniający: g, Y0 i V0y oraz obliczający na ich podstawie wielkości: t_Ymax, tz, Vyz i Ymax, według wzorów:

$$t_{Y_{max}} = \frac{V_{0Y}}{g} \qquad t_{Ziemia} = t_{Y_{max}} + \frac{\sqrt{V_{0y}^2 + 2 \cdot g \cdot Y_0}}{g}$$

$$Y_{Max} = Y\left(t_{Y_{max}}\right) \qquad V_{Y,Ziemia} = V_Y\left(t_{Ziemia}\right)$$

Zakoduj tu na nowo metody Status (z adekwatnymi monitami) i Y oraz Vy, zwracające odpowiednio:

$$Y(t) = Y_0 + V_{0Y} \cdot t - \frac{g t^2}{2} \qquad V_Y(t) = V_{0Y} - g \cdot t$$

Przejdź do konstrukcji struct Ukosny jako dziedziczącej po Pionowy. Tu nowymi polami są: V0x (składowa X prędkości początkowej) , Xmax (składowa X w chwili najwyższej wysokości) , Xz (zasięg wzdłuż osi X), Vxz (składowa X prędkości przy zderzeniu z Ziemią) . Tym razem konstruktor powinien przyjmować od użytkownika 4 pola: g, Y0, V0x i V0y, a wypełnić powinien: y_Ymax, Xmax, Ymax, tz, Vxz, Vyz i Xz. Odpowiednie wzory to:

$$t_{Y_{max}} = \frac{V_{0Y}}{g} \qquad t_{Ziemia} = t_{Y_{max}} + \frac{\sqrt{V_{0y}^2 + 2 \cdot g \cdot Y_0}}{g}$$

$$Y_{Max} = Y\left(t_{Y_{max}}\right) \qquad V_{Y,Ziemia} = V_Y\left(t_{Ziemia}\right)$$

$$X_{Ziemia} = X\left(t_{Ziemia}\right) \qquad V_{X,Ziemia} = V_X\left(t_{Ziemia}\right)$$

Zakoduj na nowo metody Status (z adekwatnymi monitami) i Y oraz Vy oraz dopisz metody X i Vx, zwracające odpowiednio:

$$X(t) = V_{0X} \cdot t \qquad V_X(t) = V_{0X}$$

$$Y(t) = Y_0 + V_{0Y} \cdot t - \frac{g t^2}{2} \qquad V_Y(t) = V_{0Y} - g \cdot t$$

Poza klasami zakoduj funkcję void PrintStatus, która przyjmie referencję na struct bazowy i wykona na nim metodę Status.

Minimalna funkcja main do testowania może wyglądać tak:

```
int main () {
  float g = 9.81;
  Swobodny S (g, 2.)         ;  PrintStatus (S);
  Pionowy  P (g, 2., 2.)     ;  PrintStatus (P);
  Ukosny   U (g, 2., 2., 2.);  PrintStatus (U);
}
```

6. Ściągnij pliki z zestawieniami dla 5 uniwersytetów: UAM_Poznan.dat, UJagiellonski.dat, ULodzki.dat, UWarszawski.dat, UWroclawski.dat. Mają one identyczną strukturę, jak np.:

Uniwersytet Warszawski
Studenci 37000
Doktoranci 1700
Nauczyciele 4100

W tym pliku masz zaczyn kodu. Ma on:

- wzorzec struct Uni. Jego obiekty mają mieścić całość informacji o danej uczelni, w polach: string Name oraz int NNaucz, NDokt, NStud.
  Ma też mieć konstruktor 4-argumentowy do ustawiania pól.

- wzorzec struct Analizator.
  Ma on pole vector<Uni> Uczelnie, do przechowywania danych o uczelniach.

Jest tu prototyp int WczytajUczelnie (string). Ta metoda ma przyjąć nazwę pliku do odczytu i otworzyć go, sprawdzając powodzenie otwarcia (w przypadku porażki, wypisz monit i zwróć -1). Następnie ma przeanalizować zawartość: wczytać nazwę uczelni i, dalej, identyfikować kategorię (Studenci, Doktoranci, Nauczyciele) i dotyczącą jej liczbę osób. (Jeśli nazwa kategorii nie pasuje, trzeba wypisać monit i zwrócić -2). Mając całość danych o uczelni, trzeba rozszerzyć vector Uczelnie o obiekt Uni z zestawem informacji o tej instytucji. Na koniec zamknij strumień i zwróć 0.

Struktura ma też prototyp int ZapiszRanking (string). Metoda ma otworzyć plik do zapisu o nazwie podanej w argumencie funkcji. Celem metody jest dokonanie rankingu uczelni pod kątem liczby studentów (od największej) i zapisanie w takiej kolejności informacji o uczelniach. Powinna dać:

```
Uniwersytet Warszawski                      37000  1700   4100
Uniwersytet Jagiellonski                    32700  1400   4950
Uniwersytet Adama Mickiewicza w Poznaniu    30700  880    3000
Uniwersytet Lodzki                          27000  380    2200
Uniwersytet Wroclawski                      22000  900    2000
```

Wskazówki, jeśli do wykonania rankingu chcesz użyć map:

- aby sortowała malejąco po kluczach, możesz zadeklarować map< ..., ..., greater<float>> ___ .
- aby uniknąć problemów z konstruktorami w Uni, do wstawiania użyj metody insert.

- Funkcja main testuje działanie Twojego algorytmu.