Programowanie Zaawansowane FM i NI
Ćwiczenia 9

Zadanie 1 (template)

Przekształć kod klasy `Regresja` w szablon (`template`).
Następnie podziel kod na plik nagłówkowy i implementacyjny szablonu oraz kod klienta.

Zadanie 2 (template)

Zmień klasy statyczne `Funkcja` i `MathTool` w szablony (`template`).
Następnie podziel kod na plik nagłówkowy i implementacyjny szablonu oraz kod klienta.

Zadanie 3 (valarray)

Kod klasy `Regresja` z zad. 7.1 przekształć tak, aby pracował na `valarray`:
– W polach zamień wskaźniki na (tablice) `double` – na referencje na obiekty `valarray<double>`. Odpowiednio zmień konstruktor (nie potrzeba argumentu `size`).
– Przepraw obliczenia zmiennych `a`, `S` i `Sa`, wykorzystując działania blokowe na obiektach `valarray`.

Uwaga: poleceniem `typedef {prawdziwy typ} {alias}` można utworzyć alias na typ, będący skrótowcem. Na przykład po poleceniu `typedef double D;` można wszędzie używać `D` jako zamiennika na typ `double`.

Zadanie 4 (`valarray<valarray<...>>, transform`)

Mała klasa 5 uczniów zbierała przez rok oceny z przedmiotu. W funkcji `main` w tym kodzie są one zestawione w formie tablic `valarray<float>`, a każda z tablic została wstawiona do obiektu `Dziennik` typu `valarray` (*uwaga*: to tablica tablic)

Twoim zadaniem jest użycie funkcji transform z biblioteki `<algorithm>`, w której dla każdego elementu tablicy `Dziennik` wywoła się funkcja `WystawOcene`. Zakoduj jej argument wejścia (unikając kopiowania). Niech ta funkcja zwróci średnią z ocen ucznia, zaokrągloną do 0.5 (przyda się `round` z biblioteki `<cmath>`).

Zadanie 5 (`valarray, pair, tuple, for_each, transform`)

Rozważ 100 punktów rozłożonych na sferze o promieniu 1 i kątach (θ, φ) o wartościach wygenerowanych losowo z zakresu $\theta = [0, \pi]$, $\varphi = [0, 2\pi]$. Twoim zadaniem jest przekształcenie tablicy stu dwuwymiarowych punktów (θ, φ) danych na tablicę stu trójwymiarowych punktów kartezjańskich [X, Y, Z]. Programistycznie zrealizuj to tak:

– Rolę tablic niech pełnią `valarray` (np. o nazwach `Angles` i `Pos`). Pojedynczym elementem tablicy `Angles` niech będzie `pair<double,double>`, a pojedynczym elementem tablicy `Pos` niech będzie `tuple<double,double,double>`.
– Wylosuj kąty w tablicy `Angles`.
– Napisz funkcję `Spher2Cart`, która pobiera parę (mieszczącą dwa kąty) i przekształca ją w 3-wymiarową tuplę ze współrzędnymi kartezjańskimi, wg transformacji:

$$[X, Y, Z] = [\, \sin\theta \cos\varphi \,,\ \sin\theta \sin\varphi \,,\ \cos\theta \,]$$

a następnie zwraca tę tuplę.
– W `main` wykorzystaj szablon `transform` z biblioteki `<algorithm>` do przeprowadzenia transformacji z `Angles` do `Pos`, przy użyciu `Spher2Cart`.
– Napisz funkcję `print`, przyjmującą 3-wymiarową tuplę i wypisującą ją na ekran.
– W `main` wykorzystaj szablon `for_each` z biblioteki `<algorithm>` do wywołania funkcji `print` na pięciu pierwszych elementach tablicy `Angles`.

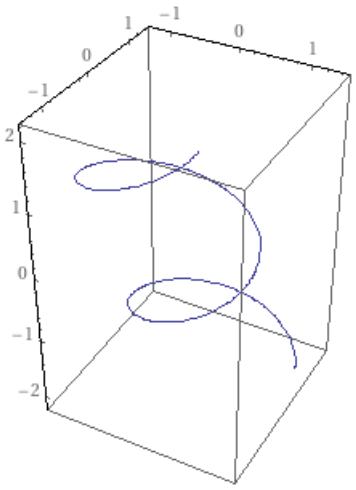
Zadanie 6 (`valarray<pair<...>>, next_permutation`)

Pasażer znajduje się na mapie w punkcie [-3, -3] i chce dotrzeć do punktu [3, 3]. Komunikacyjnie ma 4 miejsca przesiadek, w punktach [1, 2], [-1, -2] , [0, 0] i [0.5, 0.5]. Znajdź połączenie o najkrótszej drodze.

W tym kodzie punkty przesiadkowe ujęto w `valarray<pair<float,float>>` o nazwie `Points`, gdzie para odpowiada współrzędnym [X, Y] danego punktu.
- Dostępna jest funkcja `Status`, wypisująca na ekran zestaw punktów.
- Wpisana jest też funkcja `distance`, zwracająca odległość między dwoma punktami (obiektami `pair<float,float>`).

Najprostszym algorytmem jest przeiterowanie się po wszystkich permutacjach zestawu punktów przesiadkowych (dla N punktów jest ich N! , por. funkcja `silnia` w kodzie) i w każdym przypadku obliczenie łącznej drogi.
- Do nowej permutacji zbioru wykorzystaj `next_permutation` z biblioteki `algorithm`).
- Pętla po permutacjach jest rozpoczęta w funkcji `main`.
- Obliczanie łącznej drogi jest rozpoczęte w funkcji `CalculatePath`.
- Pętla powinna zawierać znajdywanie najkrótszej drogi, razem z zapamiętaniem w obiekcie `Best_arrangement` wariantu zestawu punktów, który go realizuje.

Zadanie 7 (`valarray`: działania blokowe , `min_element`, iteratory)



Naładowana cząstka porusza się w polu magnetycznym po torze helisy, tak że położenie w funkcji czasu t ma postać:

$$[X, Y, Z] = [\, cos\,(3t) - 0.2t \;,\; sin\,(3t) - 0.2t \;,\; t \,]$$

Tor dla przedziału czasów -2 < t [s] < 2 ukazuje rysunek obok. Twoim zadaniem jest wypróbkowanie położeń toru w powyższym przedziale czasów z krokiem dt = 0.01 i znalezienie punktu, w którym odległość od początku układu współrzędnych jest najmniejsza.

Spośród różnych dróg technicznego rozwiązania problemu, propozycja jest następująca:

– Obliczasz ilość punktów próbkowania `Nprob`.

– Tworzysz statyczną tablicę 3-elementową, gdzie każdym elementem jest `valarray<double>`, każdy o rozmiarze `Nprob`. Każdy z tych obiektów to tablica współrzędnych (odpowiednio) X, Y i Z kolejnych punktów.

– Wypełniasz `valarray` odpowiadający współrzędnym Z – wartościami czasu, a do wypełnienia `valarray`'ów odpowiadających X i Y używasz działań blokowych.

– Tworzysz nowy `valarray<double> Distances`, odpowiadający tablicy odległości od 0 i wypełniasz go blokowo, według wzoru na odległość punktu od [0, 0, 0].

– Używasz szablonu `min_element` z biblioteki `<algorithm>` do znalezienia iteratora na najmniejszą odległość. Przechowaj go w zmiennej o automatycznej dedukcji typu (`auto it`). Dostęp do tego elementu to: `*it`, a jego indeks uzyskasz przez: `it - begin(Distances)`.

– Wypisz znaleziony najmniejszy dystans oraz współrzędne tego punktu.