Programowanie Zaawansowane FM i NI
Ćwiczenia 10

Zadanie 1 (vector, pair, accumulate, count, count_if, predykat)

Znajdziemy przybliżenie liczby π poprzez losowanie punktów w kwadracie pomiędzy [0,0] a [1,1] (jest to tzw. metoda Monte-Carlo). Ćwiartka okręgu o promieniu 1 ma pole π/4.

W tym kodzie jest już losowanie do `valarray<pair<float,float>> Punkty`. Losowanie zachodzi poprzez `for_each` z aplikowaną funkcją `Losuj`.

1) Przepraw tablicę na pusty `vector` par, a losowanie - na osobną funkcję `Losuj`, przyjmującą cały `vector (&)` i żądaną liczbę elementów. Zastosuj `Losuj` w `main`, żądając np. 100000 punktów.
Wskazówka: `vector` poszerzamy o element metodą `push_back (element)`, a parę tworzy polecenie `make_pair( ..,.. )`.

2) Kod decyduje też, które punkty są w kole: polecenie `transform` do tablicy `Punkty` aplikuje `Decyduj`, a wyniki wpisywane są do tablicy `valarray<bool> w_kole`. Przepraw tablicę na `vector`.

3) Teraz znajdz `N_w_kole` = liczbę punktów w kole o promieniu 1. Spróbuj równolegle kilka z dróg:
   a) na tablicy `w_kole` zastosuj `accumulate` w bibliotece `<numeric>`
   b) na tablicy `w_kole` zastosuj `count` w bibliotece `<algorithm>`
   c) na tablicy `Punkty` zastosuj `count_if` w bibliotece `<algorithm>`:

```
L = count_if( iterator_pocz, iterator_za_końc , fun_predykat);
```

Musisz tu napisać *predykat*, czyli funkcję lub lambdę, która przyjmie element wektora (`pair<float,float>`), a zwróci decyzję w formie `bool`.

4) W granicy liczby losowań $N \to \infty$, $L / N = \pi / 4$. Wypisz znaleziony wynik π.

5) Usuwanie z tablicy punktów poza okręgiem. Można to zrobić nieelegancko (a, b) lub elegancko (c).
   a) przechodząc vector na piechotę i kasując element, korzystając z `erase`.
   b) funkcją `remove_if` z biblioteki `<algorithm>`, która potrzebuje predykatu. Jednak nie eliminuje ona niechcianych przypadków. Ona jedynie układa akceptowane elementy na początku, a odrzucone – na końcu. Zwraca iterator na ostatni „dobry" element:

```
vector<pair<..>>::iterator LastGood =
        remove_if ( iterator_pocz , iterator_za_końc, fun_predykat);
```

   c) (od C++20) funkcją `erase_if ( kontener, predykat )`

Uwaga 1: Kodując predykat, możesz zamiast zwykłej funkcji wpisać lambdę.

Uwaga 2: Szablon `vector` posiada własną metodę `erase`, która usuwając jeden (lub więcej) elementów, skraca rozmiar `vector`'a.

Zadanie 2 (set)

Napisz program, losujący 6 liczb naturalnych (unikalnych) z przedziału [1, 49] i wypisujący je na ekran w kolejności rosnącej.
Wskazówka: wykorzystaj cechy kontenera `set<…>` .

Zadanie 3 (deque + pair)

Dziekan ds. Studenckich rozpatruje sprawy Studentów. W chwili otwarcia biura w kolejce stało 6 osób. Wykonaj symulację ewolucji kolejki co krok czasowy odpowiadający 10 minutom, przyjmując że:
– na końcu kolejki zjawia się "regularnie" średnio R = 3 osoby / 10 minut,
– w wyjątkach "priorytetowych", przed kolejką zjawia się średnio P = 1 osoba / 10 minut,
– Dziekan rozpatruje sprawy ze średnią częstotliwością średnio U = 4.3 osoby / 10 minut.
– w każdym z powyższych przypadków liczba osób w 10 minutach podlega rozkładowi płaskiemu w przedziale [0, 2 × średnia] o średniej zadanej przez R, P lub U .

Za informację o osobie przyjmij:
– rok urodzenia (wylosowany z rozkładu jednorodnego w przedziale [1999, 2004])
– imię wylosowane jednorodnie z predefiniowanej tablicy 10 imion.

```
vector<string> names = {"Olga", "Rafal", "Jerzy", "Zofia", "Tadeusz",
                        "Anna", "Adam", "Magda", "Jozef", "Hanna"};
```

Po każdej minucie wypisuj: chwilę czasu [min], liczbę dojść regularnych i priorytetowych, liczbę obsłużeń, oraz liczebność kolejki na koniec kroku. Symulację zakończ, gdy kolejka się skończy (lub po max. 100 próbach).

Wskazówka: symulację wykonaj w oparciu o `deque` i `pair`.

Zadanie 4 (map i multimap)

W tym kodzie znajduje się lista imion zawodników z punktami za 3 różne dyscypliny sportu. Napisz program, który wypisze na ekran tabelę z wynikami zgodnie z alfabetycznym porządkiem imion. W pierwszym kroku zastosuj mechanizm `map<..,..>`. Styl może być np. taki:

```
Zawodnik    23    45    67
```

Wskazówka: kluczem wpisu może być imię (`string`), a wartością – `vector` ocen.

Uwaga: mankamentem tego mechanizmu jest wymóg unikalności imion. Aby pokonać ten problem, przejdź z `map` na `multimap<.,.>` . Jednak `multimap` nie posiada operatora `[]`. W zamian, wstawiamy nowy element na jeden z dwóch sposobów:

```
myMultiMap.insert ( {klucz, wartosc} );
```

lub

```
myMultiMap.insert ( make_pair(klucz, wartosc) );
```

Zadanie 5 (`map`, narzędzia `algorithm`)

W funkcji main tego pliku znajdują się dane o zawodnikach triathlonu: ich nazwiska, roczniki urodzenia i czasy całkowite biegów.

1) Wstaw te dane do mapy `Data`, używając pętli dowolnego rodzaju.
Dla każdego z zawodników w tej mapie wypisz ich kolejne dane. W tym celu, użyj polecenia z biblioteki `algorithm`, aby elementom zaaplikować funkcję `PrintPlayer`. Zaimplementuj tę funkcję.
Uwaga: w deklaracji `pair` będącej elementem mapy, typ klucza musi mieć przedrostek `const`.

2) Zawodnicy zdyskwalifikowani mają -1 w danej o czasie. Użyj polecenia z biblioteki `algorithm`, aby ich skasować. W tym celu zakoduj i zaaplikuj predykat `IsDisqualified` do elementów mapy `Data`.

3) Wyświetl teraz elementy mapy. Możesz np. w pętli zakresowej wywołać `PrintPlayer`.

Uwaga: Upewnij się, że kompilujesz w wystarczającym standardzie (c++20).

Zadanie 6 (`map`, `multimap`, narzędzia `algorithm`)

W funkcji main tego pliku znajduje się zestaw państw, liczb ich użytkowników internetu [mln] i ich procentowego udziału w całej populacji.

1) Wstaw te dane do mapy `Data`, używając pętli dowolnego rodzaju.
Dla każdego z miast z tej mapy wypisz ich kolejne dane. W tym celu, użyj polecenia z biblioteki `algorithm`, aby elementom zaaplikować funkcję `PrintState`. Zaimplementuj tę funkcję.

2) Utwórz sortujący kontener asocjacyjny `Data_percent` dla nieunikalnych kluczy typu `float` i wartości typu `string`. Wypełnij go parami {procent, państwo}, zgodnie z malejącym udziałem procentowym. Następnie zaimplementuj i użyj funkcji `Print_DataPercent`, aby wypisać z tego kontenera państwa i ich udziały procentowe.
Uwaga: kolejność malejącą można uzyskać albo wpisując procenty × -1 , albo korzystając z trzeciego argumentu w deklaracji szablonu:

```
_______ < ___ , ___ , greater<float> > Data_percent ;
```

3) Następnie użyj funkcji z biblioteki `algorithm`, aby sprawdzić na mapie `Data`, czy we wszystkich państwach udział przekracza 90%. W tym celu zaimplementuj predykat `IsPercOver90`. Jeżeli warunek nie jest prawdziwy, to za pomocą funkcji z biblioteki `algorithm` zaaplikuj elementom kontenera funkcję `PrintOver90`, która wypisze państwo wtedy, gdy udział użytkowników > 90%.

4) W kodzie jest kopia mapy `Data`, o nazwie `Data_Over100mln`. Użyj funkcji z biblioteki `algorithm`, aby usunąć z `Data_Over100mln` państwa, w których liczba użytkowników nie przekracza 100 mln.

Zadanie 7 (`map, multimap`)

W systemach unixowych można – pod użytkownika piszącego z klawiatury – podstawić plik tekstowy. Ściągnij ten plik z inwokacją do Pana Tadeusza. Podstawienie go „jako dane z klawiatury" wykonuje się na etapie wywołania aplikacji:

```
$ ./a.out <inwokacja.txt
```

i wówczas kolejne dane dostępne są np. przez wywołanie pętli

```
while (cin >> zmienna) {
  ...
}
```

Na końcu listy, próba przeczytania danej, której nie ma, poskutkuje zwróceniem `false` – i w efekcie pętla się skończy.

Twój program ma „wchłaniać" kolejne wyrazy i budować tablicę frekwencji słów o tej samej treści (na końcu - w % wszystkich słów) . Skorzystaj z tym celu z `map<string,int>` .

Twoim następnym celem jest posortowanie słów pod kątem malejącej frekwencji, a następnie wypisanie listy jako [słowo frekwencja%] . Możesz w tym celu utworzyć `multimap` , gdzie kluczem będzie frekwencja. Powtarzalność klucza w multimapie jest istotna, gdyż mogą być słowa z tą samą frekwencją.

Na końcu podziel zadania na funkcje: `Histogram_freq, Sortowanie_po_freq i Wypisz_wyniki`. Kontenery możesz przenosić poprzez odpowiednie referencje.