Programowanie Zaawansowane FM i NI
Ćwiczenia 12

Zadanie 1 (ifstream, ofstream, getline, argumenty wejścia main)

Napisz program, który czyta linie pliku tekstowego (może to być właśnie ten kod) i zapisuje je do pliku wyjściowego (np. o nazwie output.dat). Jednak gdy linia zaczyna się od ustalonego znaku (np. może to być #), takiej linii nie zapisuj.

Dodaj sprawdzanie powodzenia otwarcia pliku do czytania (w przypadku porażki, wypisz komunikat i zakończ program).

Przepraw teraz kod, aby nazwę pliku podawał użytkownik w pierwszym argumencie wywołania aplikacji, a wspomniany znak – w drugim. W przypadku niepodania znaku, program powinien wypisywać do pliku wszystkie linie.

Zadanie 2 W tym pliku danych ułamki zaznaczone są przecinkami. Aby był on dobrze czytany przez strumień do zmiennych typu double, trzeba je przeprawić na kropki. Napisz kod, który:

– otworzy i będzie czytał te dane linia po linii. Wskazówka: `getline (strumien, string)`
– we wczytanej linii przeprawi każdy przecinek na kropkę. (wskazówka: warto zrobić pętlę po znakach linii).
– przeprawione linie będzie zapisywał w nowym pliku.

Zadanie 3 (ifstream, istringstream, stdin)

Napisz program wypisujący na standardowe wyjście (`cout`) n-te słowo każdej linii pliku czytanego przez strumień plikowy. (Czytanym plikiem może być np. plik z Twoim kodem). Jeżeli dana linia zawiera mniej niż n słów, wypisywana powinna być linia pusta.

Następnie zmień kod tak, aby czytał on plik ze standardowego wejścia.

Zadanie 4 (stdin/stdout, ifstream/ofstream, istringstream)

Począwszy od bieżącego semestru barek sprzedaje studentom dania na kredyt, a następnie przesyła e-mailem plik z rachunkiem, np. taki. Cena jest ostatnią pozycją w linii.

Napisz program obliczający na podstawie tego pliku całkowitą należność.

W pierwszej wersji plik z rachunkiem powinien być wczytywany ze standardowego wejścia, a wynik wypisywany na standardowe wyjście.

Następnie zmień kod tak, aby czytał i zapisywał poprzez strumienie plikowe.

Zadanie 5 (ifstream, ofstream, istringstream, vector<vector<..>>, iomanip)

Współpracownicy przesyłają Ci dane doświadczalne, a wśród nich ten plik. Przyjmij, że liczbę wierszy i kolumn program pozna dopiero w efekcie czytania kodu. Wiadomo jednak, że każda kolumna ma swój jednowyrazowy nagłówek.

Twoim zadaniem jest wychwycenie takich kolumn, w których wszystkie wpisy są zerami, a następnie zapisanie pliku wyjściowego bez tych kolumn. Nazwy plików wpisz na trwałe w kod. W odczycie warto wspomóc się strumieniem napisowym. Przechowując dane, możesz np. skorzystać z `vector<vector<..>>`. Plik wyjściowy powinien prawidłowo zapisywać też nagłówki kolumn.

Następnie zmień kod tak, aby nazwy plików podawał użytkownik w opcjach wywołania. Korzystając z biblioteki `<iomanip>` , doprecyzuj format wyjścia tak, aby:
(1) wszystkie liczby zapisane były w formacie `scientific` z 4 cyframi znaczącymi,
(2) kolumny miały szerokość 10 znaków i były oddzielone pojedynczymi spacjami,
(3) ani przed pierwszą, aby za ostatnią kolumną nie były wypisywane zbędne spacje.

Zadanie 6 Skompiluj i wykonaj ten kod, który wygeneruje plik `exp_czastki.dat`, w którym znajdą się "dane eksperymentalne" ze zderzeń jąder atomowych, w rzeczywistości wylosowane. Każda linijka – to osobne zderzenie (event), a w każdym evencie eksperyment zarejestrował kilka (nieznana ilość) cząstek, których masy (w [GeV/c^2]) podaje plik. Przemyśl strukturę tego pliku.

Otrzymujesz powyższy plik do analizy. Podziel pracę na zadania:

1) napisz funkcję:

```
int Czytacz (string fileName, vector< vector <double> >& vTable)
```

czytającą plik o nazwie `fileName` i tworzącą z niego macierz danych `vTable` typu:
{ rzędy = eventy , kolumny = cząstki w danym evencie }
Sprawdzaj, czy dało się otworzyć plik: jeśli nie, to wypisz monit o błędzie i zwróć -1.
Jeśli tak – na końcu funkcji zwróć 0.
Ponieważ nie wiadomo z góry, ile cząstek jest w evencie, skorzystaj z `istringstream`.

2) Napisz funkcję:

```
void Analizator ( vector<vector<double>>& vTable )
```

która pobierze macierz danych i w każdym evencie dokona zliczenia ilości protonów i pionów. W tym celu załóż, że cząstka o masie < 0.5 to pion, a o masie > 0.5 to proton. Na koniec wypisz na ekran, ile w evencie było: cząstek, pionów i protonów.

(c.d. na nast. str.)

3) Napisz funkcję:

```
int FiltrPionow (vector<vector<double>>& vTable, string fileName)
```

której zadaniem będzie przesianie macierzy i wypisanie do pliku `fileName` linii odpowiadających eventom, gdzie w każdej linii będą tylko piony, ale tylko pod warunkiem, że w evencie znalazły się dokładnie 3 piony.

Ponieważ o liczbie pionów w evencie dowiadujemy się na końcu analizy linii, to wykorzystaj obiekt klasy istringstream jako tymczasowo formowany napis. Decyzję, czy ten tymczasowy napis zapiszesz do pliku, podejmuj po analizie każdej linii.

Zadanie 7 (strumienie plikowe w klasie)

Przekształć ten kod z klasą do obliczania regresji z danych na bazie obiektów `valarray<..>` (zad. 7.1) :

(a) Poza klasą dodaj `ostream& operator<< (ostream& Str, Regresja& R)` , która ma wykonać to samo, co metoda `Status`. Pamiętaj, że ponieważ pola klasy `Regresja` są prywatne, to w klasie trzeba użyć `friend` , aby dopuścić ten operator do pól klasy. W funkcji `main` wywołaj wypisanie statusu przez powyższy operator, zamiast przez metodę `Status`.

(b) Dodaj metodę `Wczytaj (const string& inputFileName)` , czytającą np. te dane z pliku. Niech kod sprawdza, czy otwarcie się powiodło (jeśli nie, to wyświetl komunikat i przerwij program, np. przez `exit(-1)`). Nie zakładaj, że wiadomo z góry, ile danych ma plik (*wskazówka*: warto dane wczytać wpierw do `vector`'a, a następnie zmienić go na `valarray`).

W funkcji `main` , po utworzeniu obiektu, wywołaj metodę `Wczytaj`. Jednak niech nazwę pliku podaje użytkownik w pierwszej opcji wywołania kodu.

(c) Dodaj metodę `void Zapisz (const string& inputFileName)`, w której oprogramujesz zapis wyniku do pliku o zadanej nazwie (np. tak, jak wypis w metodzie `Status`). Sprawdzaj, czy nie ma problemu z otwarciem pliku (jeśli nie, to wyświetl komunikat i przerwij program, np. przez `exit(-1)`).

W funkcji `main` , po utworzeniu obiektu, wywołaj metodę `Zapisz`. Nazwę pliku niech podaje użytkownik w drugiej opcji wywołania kodu.

(c.d. na nast. str.)

Zadanie 8 (strumienie plikowe w klasie)

W tym kodzie znajdziesz implementację minimalną zagadnienia „Notes Osób o informacji imię+telefon” w formie klasy `Notes`, w której polem danych jest prywatna `map<int,string> Data`. Prosty konstruktor służy do przyjęcia zbioru par {imię, telefon}, co jest wykorzystane w funkcji `main`. W kodzie znajdziesz też metodę `Wypisz`, wypisującą na ekran zawartość notesu.

Rozwiń kod w następujący sposób:

– dodaj konstruktor `Notes (string fileName)`, który otworzy plik z danymi i sprawdzi powodzenie otwarcia (jeśli nie, niech przerwie kod przez `exit (1);`), a następnie wczyta dwukolumnowe dane do mapy. Zamknij strumień. W funkcji `main` zmień inicjalizację obiektu klasy `Notes` na taką, które używa tego (nowego) konstruktora. Jako dane, wpisz wpierw nazwę tego pliku.

– Zamień metodę `Wypisz` na `operator<<` pomiędzy strumieniem wyjściowym a obiektem klasy `Notes`. Ma on wypisywać to samo, co `Wypisz`, tylko do dowolnego strumienia wyjściowego. Ponieważ operator ten jest poza klasą, a w klasie pole z danymi jest prywatne, to potrzeba dopuścić `operator<<` poprzez `friend`. W funkcji `main` podmień metodę `Wypisz` na `cout << N`. Następnie otwórz strumień zapisowy plikowy, podając nazwę np. `notes.out`. Stosując ten sam operator, wypisz zawartość notesu do tego pliku. Zamknij strumień.

– Rozszerz `main` o argumenty wejścia i zmień ciało tej funkcji tak, aby tworzony obiekt klasy `Notes` pobierał nazwę pliku z pierwszego argumentu wejścia. Przy otwarciu strumienia wyjściowego do zapisu, niech nazwa pliku będzie pobrana z drugiego argumentu wejścia funkcji `main`.