Zadanie $\gamma 1$. Vector, 0.5 pkt

Celem zadania jest zaimplementowanie własnej klasy `vector` realizującej tablicową implementację listy. Definicję klasy wraz z definicjami wszystkich metod i operatorów należy umieścić w pliku z rozszerzeniem `.h`. Tak przygotowany plik należy wysłać na Satori. Zostanie on skompilowany wraz z plikiem zawierającym funkcję `main`.

Klasa `vector`

1. Posiada trzy pola prywatne:
   - `tab` - tablica wartości typu `string`
   - `capacity` – rozmiar tablicy `tab`
   - `size` – aktualna liczba elementów przechowywana w tablicy `tab`
2. Posiada dwa konstruktory:
   - z trzema parametrami: `int a`, domyślnie ustawionym na 4, `int b` domyślnie ustawionym na 0 oraz `string c` domyślnie ustawionym na "" – tworzy obiekt przechowujący maksymalnie a wartości i wstawia do początkowych b komórek tablicy wartość c.
   - konstruktor kopiujący
3. Udostępnia następujące metody:
   - `add(string x)` – dodaje element `x` na koniec `tab`. Jeśli tablica jest pełna, funkcja dwukrotnie zwiększa tablicę i dodaje element `x`.
   - `insert(int i, string x)` – dodaje element x na pozycję i-tą (pozostałe elementy są przesuwane). Pozycje liczone są od 0. Jeśli pozycja jest większa od `size`, funkcja nie robi nic. Jeśli tablica jest pełna, funkcja dwukrotnie zwiększa tablicę i wstawia element `x` na pozycję i.
   - `erase(int i)` – usuwa element z pozycji i-tej (pozostałe elementy są przesuwane). Jeśli pozycja jest niepoprawna, funkcja nie robi nic.
   - `getSize()` – zwraca liczbę elementów przechowywanych w tablicy `tab`.
   - `resize(int s)` – ustawia liczbę elementów przechowywanych w tablicy na s. Jeśli trzeba, funkcja zwiększa rozmiar tablicy przemnażając go przez odpowiednią wielokrotność liczby 2. Elementy dodatkowe ustawia na "".
   - `clear()` – ustawia liczbę elementów przechowywanych w tablicy na 0.
4. Przeładowane są następujące operatory
   - `operator[]` – operator dostępu do tablicy. Jeśli zadany jako parametr indeks jest większy lub równy `size`, funkcja zwraca referencję do ostatniego elementu, czyli elementu znajdującego się pod indeksem `size-1`. Można założyć, że operator nie będzie wywoływany na wektorze pustym.

- `operator=` – operator przypisania
- `operator<<` – operator wypisywania, dla zadanego obiektu `A` wypisuje po spacji `A.size` wartości typu `string` z tablicy.
- `operator>>` –operator wczytywania, dla zadanego obiektu `A` wczytuje `A.size` wartości typu `string`.

Dostępna pamięć: w zależności od testu 2-110MB
Wymagany język: C++

Przykład

Przykładowy plik z funkcją `main`:

```
#include<iostream>
#include<string>
using namespace std;
#include "solution.h"

void insertionSort(vector tab[], int n) {
   for(int a = 1; a < n; a++)  {
      vector c = tab[a];
      int x = a;
      for(;x > 0 && c[0] < tab[x-1][0]; x--)  {
         tab[x] = tab[x-1];
         tab[x-1] = c;
      }
   }
}

int main() {
   ios_base::sync_with_stdio(false);
   vector data;

   for(int i=0; i<8; i++) data.add("kkk");
   for(int i=0; i<8; i++) cout << data[i] << " ";
   cout << endl;

   vector data2(80);
   data2.resize(8);
   data2[0] = "a";
   data2[1] = "A";
   data2[2] = "b";
   data2[3] = "B";
   data2[4] = "c";
```

```
data2[5] = "d";
data2[6] = "e";
data2[7] = "f";

cout << data2 << " " << data2.getSize() << endl;
data2.insert(11, "ala");
data2.insert(8, "koza");
data2.insert(5, "Viola");
cout << data2 << " " << data2.getSize() << endl;

vector data3(data2);
data3.erase(13);
data3.erase(5);
cout << data3 << " " << data3.getSize() << endl;
cout << data3[5] << endl;

vector data4(200, 5, "Karolina");
cout << data4 << " " << data4.getSize() << endl;
cin >> data4;
cout << data4 << " " << data4.getSize() << endl;
cin >> data4;

data4 = data;
cout << data4 << " " << data4.getSize() << endl;
data4.clear();
data4.insert(0, "worek");
cout << data4 << " " << data4.getSize() << endl;

data2 = data2;
data2[0] = "Tatry";
cout << data2 << " " << data2.getSize() << endl;

vector data5 = data2;
data5.resize(5);
data5[0] = "abecadlo";
cout << data5 << " " << data5.getSize() << endl;

vector tab[6];
tab[0] = data;
tab[1] = data2;
tab[2] = data3;
tab[3] = data4;
tab[4] = data5;

tab[1][0] = "mysza";
tab[2][0] = "kot";
```

```
   insertionSort(tab, 5);
   cout << tab[0] << endl;
   cout << tab[1] << endl;
   cout << tab[2] << endl;
   cout << tab[3] << endl;
   cout << tab[4] << endl;
   cout << data2 << endl;
   return 0;
}
```

Wynik działania powyższej funkcji `main` dla danych wejściowych `ala kot Ala KOT Krowa`:

```
kkk kkk kkk kkk kkk kkk kkk kkk
a A b B c d e f  8
a A b B c Viola d e f koza  10
a A b B c d e f koza  9
d
Karolina Karolina Karolina Karolina Karolina  5
ala kot Ala KOT Krowa  5
kkk kkk kkk kkk kkk kkk kkk kkk  8
worek  1
Tatry A b B c Viola d e f koza  10
abecadlo A b B c  5
abecadlo A b B c
kkk kkk kkk kkk kkk kkk kkk kkk
kot A b B c d e f koza
mysza A b B c Viola d e f koza
worek
Tatry A b B c Viola d e f koza
```

Zadanie $\gamma 2$*. Sortowanie kubełkowe, 0.5 pkt

Danych jest n napisów (ciągów zawierających duże litery alfabetu angielskiego). Napisz program, który uporządkuje zadane napisy w kolejności niemalejącej stosując metodę **sortowania kubełkowego**.

W rozwiązaniu zastosuj klasę `vector` z zadania $\gamma 1$, dla której zdefiniuj dodatkowo rekurencyjną metodę `sort(int i)` wykorzystującą sortowanie kubełkowe, w którym *kubełki* odpowiadają literom alfabetu. Funkcja sortuje według i-tej litery napisu. Dodatkowo możesz zaimplementować metodę `insertSort()` realizującą sortowanie przez wstawianie i wykorzystywaną do posortowania tablicy napisów, gdy jest ich mało (np. mniej niż 30).

Wejście

Pierwsza linia wejścia zawiera liczbę całkowitą z ($1 \leqslant z \leqslant 2 \cdot 10^9$) – liczbę zestawów danych, których opisy występują kolejno po sobie. Opis jednego zestawu jest następujący:

Pierwsza linia zestawu zawiera jedną liczbę naturalną n ($1 \leqslant n \leqslant 20000$) — liczbę napisów do posortowania. Druga linia zawiera n oddzielonych spacjami napisów (ciągów zawierających duże litery alfabetu angielskiego.

Wyjście

Dla każdego zestawu danych, wypisz wczytane napisy w kolejności niemalejącej.

Dostępna pamięć: w zależności od testu 2-110MB
Wymagany język: C++

Przykład

Dla danych wejściowych:

```
2
8
MICKIEWICZ KOCHANOWSKI GOMBROWICZ KOSSAK KRASINSKI KOSSOWSKI MILKOWSKI GONIEWSKI
30
AJ AI AH AG AF AE AD AC AB AA J I H G F E D C B A AAJ AAI AAH AAG AAF AAE AAD AAC AAB AAA
```

Poprawną odpowiedzią jest:

```
GOMBROWICZ GONIEWSKI KOCHANOWSKI KOSSAK KOSSOWSKI KRASINSKI MICKIEWICZ MILKOWSKI
A AA AAA AAB AAC AAD AAE AAF AAG AAH AAI AAJ AB AC AD AE AF AG AH AI AJ B C D E F G H I J
```