# Zadanie G
# Bajto-Lotek

Bajtocjanie uwielbiają grać w Bajto-Lotka. Dla niektórych z nich, kupowanie losu stało się rytuałem każdego poranka oprócz picia BajtoKawy i czytania Bajterii.

Kupon Bajto-Lotka składa się z ciągu 10 cyfr. Nabywca kuponu, aby wziąć udział w losowaniu musi zarejestrować swój kupon w kiosku Bajto-Lotka lub za pomocą serwisu internetowego. Podczas rejestracji podaje on ciąg cyfr z kuponu oraz swoje nazwisko. Cechą narodową Bajtocjan jest częste zmienianie decyzji. Dlatego też serwis Bajto-Lotka umożliwia dodatkowo wycofanie zarejestrowanego kuponu i ponowne go zarejestrowanie innego dnia.

Każdego dnia maszyna losująca wybiera pewien ciąg 10 cyfr `p`. Zwycięzcami losowania są: Bajtocjanin, którzy zarejestrował kupon o wartości `p` (o ile taki jest) oraz dwaj dodatkowi Bajtocjanie tacy, że wartości ich kuponów są odpowiednio poprzednikiem i następnikiem kuponu o wartości `p` w kolejności leksykograficznej kuponów. Wybrane osoby dzielą się po równo pewną kwota Bajtodukatów i nadal mogą brać udział w w losowaniach.

W każdy poniedziałek w Bajto-Lotku obowiązują inne zasady. Tego dnia do podziału jest większa kwota Bajtodukatów i losowane są dwa kupony `k1` i `k2`. Zwycięzcami są wszystkie osoby, których kupony są pomiędzy wylosowanymi wartościami.

Zostałeś poproszony o napisanie fragmentu oprogramowania do obsługi Bajto-Lotka. W szczególności Twój program powinien obsługiwać następujące operacje:

- `INSERT kupon nazwisko` - dodaje nowy kupon. Jeśli kupon jest już w bazie operacja zmienia przypisane mu nazwisko.
- `FIND kupon` - wypisuje w jednej linii nazwisko Bajtocjanina, którzy zarejestrował kupon `kupon` poprzedzone słowami: `FIND kupon`. Jeśli w bazie nie ma takiego kuponu należy wypisać `FIND ERROR`.
- `DELETE kupon` - usuwa z bazy wskazany kupon. Wypisuje `OK` jeśli operacja wykonała się poprawnie lub `ERROR`, jeżeli nie ma wskazanego kuponu.
- `PREV kupon` - wypisuje jeden kupon (jego wartość wraz z nazwiskiem Bajtocjanina) największy leksykograficznie, ale mniejszej niż `kupon`. Jeżeli nie ma takiego kuponu, operacja wypisuje `ERROR`.
- `NEXT kupon` - wypisuje jeden kupon (jego wartość wraz z nazwiskiem Bajtocjanina), najmniejszy leksykograficznie, ale większy niż `kupon`. Jeżeli nie ma takiego kuponu, operacja wypisuje `ERROR`.
- `PRINT` - wypisuje osoby z bazy kuponów (kupony i nazwiska) w porządku leksykograficznym względem wartości `kupon`. Jeśli baza jest pusta, operacja wypisuje `EMPTY`.
- `SIZE` - wypisuje liczbę zarejestrowanych kuponów.
- `FIRST` - wypisuje najmniejszy leksykograficznie zarejestrowany kupon. Jeśli baza jest pusta, operacja wypisuje `EMPTY`.
- `LAST` - wypisuje największy leksykograficznie zarejestrowany kupon. Jeśli baza jest pusta, operacja wypisuje `EMPTY`.

- `MONDAY k1 k2` - wypisuje osoby z bazy kuponów (kupony i nazwiska), takie że wartość `kupon` $\in [k1, k2]$ w porządku leksykograficznym ($k1 \leqslant k2$). Jeśli baza jest pusta, operacja wypisuje `EMPTY`. Wypisywanie poprzedza linią: `MONDAY k1 k2`.

Zadanie należy zrealizować przez zaimplementowanie szablonów trzech klas: `node<T1,T2>`, `map<T1,T2>` oraz `iterator<T1,T2>`. Szablon `map` realizuje słownik, czyli posortowany kontener asocjacyjny o zmiennej długości. Elementami słownika są unikatowe pary klucz i element (typu `T1` i `T2`). Dostęp do elementów słownika uzyskujemy za pomocą iteratorów, czyli obiektów klasy `iteratorM<T1,T2>`, które wskazują na elementy klasy `map<T1,T2>`.

Słownik `map` należy zaimplementować za pomocą **reprezentacji wskaźnikowej drzewa przeszukiwań binarnych (BST)**, którego elementami są obiekty klasy `node`. Nie jest wymagane równoważenie drzewa.

1. **Klasa** `node<T1,T2>` przechowuje pary o typach `T1` i `T2` oraz udostępnia następujące metody:
   - `node(T1 k,T2 n)` – konstruktor, nowy obiekt przechowuje wartości `k` oraz $n$.
   - `ostream& operator<<(ostream& st, node<T1,T2>& a)` – operator wypisuje wartości obiektu a w jednej linii oddzielając je spacją.
   - `T1 GetKey()` – zwraca przechowywane pole typu `T1`.
   - `T2 GetName()` – zwraca przechowywane pole typu `T2`.
2. **Klasa** `map<T1,T2>` udostępnia następujące metody:
   - konstruktor tworzący pusty słownik
   - `void insert(T1 key, T2 name)` – dodaje do słownika parę `(key,name)`. Jeśli w słowniku jest para o kluczu `key`, operacja ją modyfikuje.
   - `bool erase(T1 key)` – usuwa ze słownika parę o kluczu `key`. Zwraca `true`, gdy usuwanie się powiodło, `false`, gdy nie ma w słowniku pary o wskazanym kluczu.
   - `bool erase(iteratorM<T1,T2> it)` – usuwa ze słownika element wskazywany przez iterator `it`. Zwraca `true`, gdy usuwanie się powiodło, `false`, gdy `it` wskazuje na `nullptr`.
   - `iteratorM<T1,T2> find(T1 key)` – zwraca iterator do elementu o kluczy `key`. Gdy elementu nie ma, metoda zwraca iterator wskazujący na `nullptr`.
   - `iteratorM<T1,T2> findOrNext(T1 key)` – zwraca iterator do elementu o kluczy `key`. Gdy takiego elementu nie ma, metoda zwraca iterator do elementu o najmniejszym kluczu większym od `key`. Gdy takiego elementu również nie ma, metoda zwraca iterator wskazujący na `nullptr`.
   - `iteratorM<T1,T2> begin()` – zwraca iteratora do pierwszej pary w słowniku. Gdy słownik jest pusty, metoda zwraca iterator wskazujący na `nullptr`.
   - `iteratorM<T1,T2> end()` – zwraca iterator do ostatniej pary w słowniku. Gdy słownik jest pusty, metoda zwraca iterator wskazujący na `nullptr`.
   - `void print()` – wypisuje wszystkie pary ze słownika posortowane według kluczy.

- `void print(T1 k1, T1 k2)` – wypisuje wszystkie pary słownika posortowane według kluczy takie, że klucz pary jest większy lub równy `k1` oraz mniejszy lub równy `k2`.
- `int size()` – zwraca liczbę par w słowniku.
- `void clean()` – czyści słownik.
- `T2& operator[](T1 key)` – operator zwraca referencję do pola typu `T2` pary o kluczu `key`. Jeśli takiej pary nie ma w słowniku, zostaje dodana.

3. **Iterator** `iteratorM<T1,T2>` udostępnia następujące funkcjonalności:

- `iteratorM(node<T1,T2>* a)` - konstruktor, nowy iterator wskazuje na obiekt wskazywany przez a.
- konstruktor kopiujący
- `node<T1,T2>& operator*()` – zwraca obiekt wskazywany przez iterator.
- `node<T1,T2>* operator->()` – zwraca wskaźnik na obiekt wskazywany przez iterator.
- `operator bool()` – zwraca wartość `true` gdy iterator wskazuje na obiekt różny od `nullptr`, w przeciwny razie zwraca `false`.
- `iteratorM<T1,T2> operator++(int)` – operator modyfikuje iterator, który teraz wskazuje na następną parę w słowniku. Zwraca nowo wskazywaną parę. Jeśli takiej pary nie ma lub iterator wskazywał na `nullptr`, zwraca iterator wskazujący na `nullptr`.
- `iteratorM<T1,T2> operator--(int)` – operator modyfikuje iterator, który teraz wskazuje na poprzednią parę w słowniku. Zwraca nowo wskazywaną parę. Jeśli takiej pary nie ma lub iterator wskazywał na `nullptr`, zwraca iterator wskazujący na `nullptr`.

Definicję wymienionych szablonów 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`.

## 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 zawiera liczbę naturalną $n$ ($1 \leqslant n \leqslant 2 \cdot 10^6$) oznaczającą ilość operacji do wykonania. Kolejne $n$ linii zawiera: kod operacji oraz stosowne dla operacji argumenty oddzielone spacja. Argumentami są (w zależności od operacji): kupon (10-znakowy napis składający się z cyfr) oraz nazwisko (maksymalnie 10 znakowy napis złożony z małych liter alfabetu angielskiego).

## Wyjście

Każdą wczytaną operację wykonaj zgodnie z jej opisem.

**Wersja G1** - nie obsługuje poleceń: `PREV`, `NEXT`, `MONDAY`, metod: `findOrNext(string)`, `print(T1, T1)`, operatora `++` oraz operatora `--`, wersja za 1 pkt.
**Wersja G2*** - obsługuje wszystkie polecenia, wersja za dodatkowe 0.5 pkt.

Przykładowy plik z funkcją `main`:

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

int main()
{
   ios_base::sync_with_stdio(false);
   int z, n;
   string coupon, coupon2, name;
   map<string, string> T;

   cin >> z;
   while(z--)
   {
      cin >> n;
      for (int i=0; i<n; ++i)
      {
        string op;
        cin >> op;

        if (op == "INSERT")
        {
           cin >> coupon >> name;
           T[coupon] = name;
        }

        if (op == "FIND")
        {
            cin >> coupon;
            iteratorM<string, string> it = T.find(coupon);
            cout << "FIND ";
            if (it) cout << it->GetKey() << " " << it->GetName() << endl;
            else cout << "ERROR" << endl;
        }

        if (op == "DELETE")
```

```
        {
            cin >> coupon;
            if (T.erase(coupon)) cout << "OK" << endl; else cout << "ERROR" << endl;
        }
        if (op == "PRINT")  T.print();
        if (op == "SIZE") cout << T.size() << endl;
        if (op == "FIRST")
        {
            iteratorM<string, string> it = T.begin();
            if (it) cout << *it; else cout << "EMPTY" << endl;
        }
        if (op == "LAST")
        {
            iteratorM<string, string> it = T.end();
            if (it) cout << *it; else cout << "EMPTY" << endl;
        }
        if (op == "PREV")
        {
           cin >> coupon;
            iteratorM<string, string> it = T.findOrNext(coupon);
            if (!it) it = T.begin();  else it--;
            if (it) cout << *it; else cout << "ERROR" << endl;
        }
        if (op == "NEXT")
        {
           cin >> coupon;
            iteratorM<string, string> it = T.findOrNext(coupon);
            if ((it) && (it->GetKey() == coupon)) it++;
            if (it) cout << *it; else cout << "ERROR" << endl;
        }
        if (op == "MONDAY")
        {
           cin >> coupon >> coupon2;
           cout << "MONDAY " << coupon << " " << coupon2 << endl;
           T.print(coupon, coupon2);
        }
     }    //for

     T.clean();
   }
   return 0;
}
```

**Dostępna pamięć: w zależności od testu 2-16MB**

## Przykład

Dla danych wejściowych:

```
1
33
INSERT 0123456789 kowalski
INSERT 3321000000 iglinski
INSERT 9220123456 bednarski
INSERT 8220123456 kwaklinski
INSERT 0000067890 nowak
INSERT 0000667890 kowalski
INSERT 2321000000 abacki
PRINT
INSERT 9220123456 nowak
FIND 9220123456
FIND 0000067890
FIND 0000000890
FIND 3321000000
PREV 3321000000
PREV 3321000001
NEXT 3321000000
NEXT 3321000001
PREV 0000067890
NEXT 9220123456
PREV 0123456789
NEXT 0123456789
MONDAY 0000667880 3321000000
DELETE 0000000000
DELETE 3321000000
PRINT
DELETE 0123456789
PRINT
FIND 0123456789
FIND 8220123456
FIND 3321000000
FIRST
LAST
SIZE
```

Poprawną odpowiedzią jest:

```
0000067890 nowak
0000667890 kowalski
0123456789 kowalski
2321000000 abacki
3321000000 iglinski
8220123456 kwaklinski
9220123456 bednarski
FIND 9220123456 nowak
FIND 0000067890 nowak
FIND ERROR
FIND 3321000000 iglinski
2321000000 abacki
3321000000 iglinski
8220123456 kwaklinski
8220123456 kwaklinski
ERROR
ERROR
0000667890 kowalski
2321000000 abacki
MONDAY 0000667880 3321000000
0000667890 kowalski
0123456789 kowalski
2321000000 abacki
3321000000 iglinski
ERROR
OK
0000067890 nowak
0000667890 kowalski
0123456789 kowalski
2321000000 abacki
8220123456 kwaklinski
9220123456 nowak
OK
0000067890 nowak
0000667890 kowalski
2321000000 abacki
8220123456 kwaklinski
9220123456 nowak
FIND ERROR
FIND 8220123456 kwaklinski
FIND ERROR
0000067890 nowak
9220123456 nowak
5
```