Databases

locked

In this challenge, the objective is to implement a `Database` class which is templated on the table schema. A `Database` class will store a collection of **records**, where each record will consist of a **key** and a **value**, which themselves are collection of fields (also called the **schema**). For example, in a student database, the key could be the student roll number (of type `string`), while the value could consist of fields like name, DOB, stream, semester, etc. The key may also consist of multiple fields, for example, in a database storing details about BTP/DDP guides of students, the key would be student roll number and faculty name. The point is that for every key instance, there would be at most one record with that key in the database.

The `Database` class will be templated on classes `Key` and `Value`. You need to create a `Record` class which stores all details of a record (i.e. the `Key` and the `Value`). The `Database` class should then store all the records using the `DoublyLinkedList` class created in challenge-1. You should then implement the following methods in the `Database` class:

- `isPresent(Key & k)` returns `true` if the input key is present in the database, otherwise `false`. `selectRecord(Key & k)` returns the record containing the key (you can assume that `k` will be present).
- `selectRangeRecord(Key & rangeStart, Key & rangeEnd)` returns a vector containing all the records in the database whose key lies between `rangeStart` and `rangeEnd` (inclusive).
- `updateRecord(Key & k, Value & v)` updates the record in the database whose key is `k` with value `v`. If `k` is not present, there is no change in the database.
- `updateRangeRecord(Key & rangeStart, Key & rangeEnd, Value & v)` updates all records in the database whose key lies between `rangeStart` and `rangeEnd`, with value `v` (inclusive).
- `insertRecord(Key & k, Value & v)` inserts a new record with `k` and `v`.
- `deleteRecord(Key & k)` deletes the record with key `k`. `deleteRangeRecord(Key & rangeStart, Key & rangeEnd)` deletes all records with key between `rangeStart` and `rangeEnd` (inclusive).
- `getMinRecord()` and `getMaxRecord()` return the minimum and maximum record respectively (you can assume that the database won't be empty).
- `getallRecords()` returns a vector containing all the records in the database.

To implement these operations efficiently, you should ensure that while adding/deleting records from the linked list, it remains sorted in increasing order according to `Key`. Now, while searching for a record with a specific `Key k`, you should implement a binary search-like procedure which should make at most $log n$ calls to `atIndex` method of `DoublyLinkedList`, where n is the number of records in the database. To summarize, use the fact that the linked list is sorted to implement the methods as efficiently as possible. Note that you should not change the DoublyLinkedList implementation from challenge-1. Your implementation of `Database` class should call appropriate methods of the `DoublyLinkedList` class. You need not worry about the time complexity of the methods in the DoublyLinkedList implementation.

There will be a 30% penalty for inefficient implementations.

Input Format

Input is handled by the starter code. Every input is terminated by a command "50". Which is terminating command.

Constraints

None.

Output Format

Output is handled by the starter code.

Sample Input 0

```
1
2
STUDENT AE19S078 Pearce Jamuna 7.57
COURSE CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
2
STUDENT AE19S078
2
FACULTY CS1
50
```

Sample Output 0

```
1
0
```

Explanation 0

Student with roll number AE19S078 is present in the student database and there is no entry in faculty database.

Sample Input 1

```
1
2
STUDENT AE19S078 Pearce Jamuna 7.57
COURSE CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
3
COURSE CS6122
3
STUDENT AE19S078
50
```

Sample Output 1

```
CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
AE19S078 Pearce Jamuna 7.57
```

Sample Input 2

```
1
5
COURSE CS6508 Natural_Language_Processing EVEN BVVR
COURSE CS6044 Speech_Technology ODD RAVI
COURSE CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
COURSE CS6480 Reinforcement_Learning EVEN YADU
COURSE CS6196 Stochastic_Optimization ODD KKR
4
COURSE CS6196 CS6480
50
```

Sample Output 2

```
CS6196 Stochastic_Optimization ODD KKR
CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
CS6480 Reinforcement_Learning EVEN YADU
```

Sample Input 3

```
1
5
COURSE CS6508 Natural_Language_Processing EVEN BVVR
COURSE CS6044 Speech_Technology ODD RAVI
COURSE CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
COURSE CS6480 Reinforcement_Learning EVEN YADU
COURSE CS6196 Stochastic_Optimization ODD KKR
5
COURSE CS6196 Multi-armed_bandits EVEN BVVR
5
COURSE CS6521 Speech_Technology ODD RAVI
3
COURSE CS6196
2
COURSE CS6521
50
```

Sample Output 3

```
CS6196 Multi-armed_bandits EVEN BVVR
0
```

Sample Input 4

```
1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
6
FACULTY CS10 CS15 Glynda Professor 30000
4
FACULTY CS10 CS15
50
```

Sample Output 4

```
CS10 Glynda Professor 30000
CS15 Glynda Professor 30000
```

Sample Input 5

```
1
5
STUDENT PH23M044 Ives Alakananda 9.58
STUDENT PH23M099 Quillan Mandakini 8.17
STUDENT PH23S025 Ailee Mahanadi 6.5
STUDENT PH23S056 Luella Mahanadi 7.3
STUDENT PH23S057 Beatrisa Mandakini 8.93
5
STUDENT PH23S025 Beatrisa Mandakini 8.93
3
STUDENT PH23S025
50
```

Sample Output 5

```
PH23S025 Beatrisa Mandakini 8.93
```

Sample Input 6

```
1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
4
FACULTY CS10 CS24
7
FACULTY CS15
4
FACULTY CS10 CS24
50
```

Sample Output 6

```
CS10 Gaston Assoc.Professor 20000
CS15 Elvin Asst.Professor 10000
CS24 Bobbe Assoc.Professor 20000
CS10 Gaston Assoc.Professor 20000
CS24 Bobbe Assoc.Professor 20000
```

Sample Input 7

```
1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
4
FACULTY CS09 CS24
8
FACULTY CS10 CS15
2
FACULTY CS15
4
FACULTY CS09 CS24
50
```

Sample Output 7

```
CS09 Darsey Professor 30000
CS10 Gaston Assoc.Professor 20000
CS15 Elvin Asst.Professor 10000
CS24 Bobbe Assoc.Professor 20000
0
CS09 Darsey Professor 30000
CS24 Bobbe Assoc.Professor 20000
```

Sample Input 8

```
1
3
STUDENT AE23D016 Bobbe Mandakini 7.82
STUDENT ME19B05 Earlie Mandakini 6.93
STUDENT AE19S078 Pearce Jamuna 7.57
9
STUDENT
50
```

Sample Output 8

```
AE19S078 Pearce Jamuna 7.57
```

Sample Input 9

```
1
3
STUDENT AE23D016 Bobbe Mandakini 7.82
STUDENT ME19B05 Earlie Mandakini 6.93
STUDENT AE19S078 Pearce Jamuna 7.57
10
STUDENT
50
```

Sample Output 9

```
ME19B05 Earlie Mandakini 6.93
```

Submissions: 85
Max Score: 60
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

C++20

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;
template<class object>
int Bsearch(object* arr, int n, int m, object k)
{

    while(n!=m)
    {
        int mid = (n+m)/2;
        if (arr[mid]<k)
        {
            n = mid+1;
        }
        else if (arr[mid]>k)
        {
            m = mid-1;
        }
        else{
            return mid;
        }
    }
    if (arr[n]>k)
    {
        return n;
    }
    else{
        return n+1;
    }
    return n;
}
template<class Object>
class DoublyLinkedList
{
private:
int lastindex;
struct Node
{
Object data;
Node *next;
Node *prev;
          Node(const Object & d = Object(), Node * p = nullptr, Node * n = nullptr)
              : data(d), prev(p), next(n) {}
};

Node *head; //sentinel node at the beginning
Node *rear; //sentinel node at the end


public:

/*TODO: Define a constructor for DoublyLinkedList here, allocating the sentinel nodes*/
        DoublyLinkedList (){
            head = new Node; rear = new Node;
            head->prev = NULL;
            rear->next = NULL;
            head->next = rear;
            rear->prev = head;
            lastindex = -1;
        }

class Iterator
{
private:
                Node *current;
public:
                Iterator() { }
                Iterator(Node *inp) : current(inp) {}
                Object & operator*() {return current->data;}
                Iterator & operator ++()
                {
                    this->current = this->current->next;
                    return *this;
                }
                Iterator & operator --()
                {
                    this->current = this->current->prev;
                    return *this;
                }
                bool operator != (Iterator rhs)
                {
                    if (this->current==rhs.current){return false;}
                    else {return true;}
                }


/*TODO: You can add more methods here */

friend class DoublyLinkedList<Object>;
```

```
//friend class Database;
};
int lstid(){return lastindex;}
Iterator begin(){return Iterator(head->next);}
Iterator end(){return Iterator(rear);}
void insert(Iterator itr, Object o)
        {
            lastindex++;
            Node* nn = new Node;
            nn->data = o;
            (itr.current)->prev->next = nn;
            nn->prev = (itr.current)->prev;
            (itr.current)->prev = nn;
            nn->next = itr.current;
        }
void erase(Iterator itr)
        {
            lastindex--;
            (itr.current)->next->prev = itr.current->prev;
            (itr.current->prev)->next = itr.current->next;
        }
Iterator atIndex(int p)
        {/* Implement here */
    //cout<<p<<endl;
           Iterator ff;
         ff = Iterator(head->next);
            for (int ii=0; ii<p; ii++)
            {
               ++ff;
            }
           return ff;

        }
int indexOf(Iterator itr)
        {/* Implement here */
            auto ptr = (begin()).current;
            for (int ii=0;ii<=lastindex;ii++)
            {
                if (ptr->data==*(itr))
                {
                    return ii;
                }
                ptr=ptr->next;
            }
            return -1;
        }
void display()
{
     for (auto it = begin(); it != end(); ++it)
     cout << *it << " ";
     cout << endl;
}
};
//template<class Key, class Value>
template <class Key, class Value>
class Database{
public:
    class Record {
    private:
        Key k;
        Value v;
    public:
        Record(const Key &ik = Key(), const Value &iv = Value()) : k(ik), v(iv) {}
        Key &getKey() { return k; }
        Value &getValue() { return v; }
        const bool & operator == (const Record & rhs)
        {
           Key t = rhs.k;
           return !(this->k < t || t < this->k);
        }
        friend class Database<Key, Value>;
    };

private:
    DoublyLinkedList<Record> list;
    /*use the DoublyLinkedList class created in challenge 1*/
public:
    Database() {DoublyLinkedList<Record> list;}
    /*TODO: Implement the methods here*/
    void insertRecord(Key k, Value v)
    {
        int n = 0, m = list.lstid();
        if (n>m)
        {
            list.insert(list.atIndex(0), Record(k, v));
            return;
        }
    while(n<m)
    {
        int mid = (n+m)/2;
        if ((*(list.atIndex(mid))).k<k)
        {
            n = mid+1;
        }
        else if (k<(*(list.atIndex(mid))).k)
        {
            m = mid-1;
        }
        else{
            return;
        }
    }
    if (k<(*(list.atIndex(n))).k)
    {
        list.insert(list.atIndex(n), Record(k, v));
        return;
    }
    else{
        list.insert(list.atIndex(n+1), Record(k, v));
        return;
    }
    }
    bool isPresent(Key k)
    {
        int hh=-1;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<k)
            {
              n = mid+1;
            }
            else if (k<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              hh = mid;
              return true;
            }
         }
         if (!(((*(list.atIndex(n))).k<k)||(k<(*(list.atIndex(n))).k))) { hh = n; return true;}
        return false;
    }
    Record selectRecord(Key k)
    {
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<k)
            {
              n = mid+1;
            }
            else if (k<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              return *(list.atIndex(mid));
            }
         }
        return *(list.atIndex(n));
    }
    vector<Record> selectRangeRecord(Key rs, Key re)
    {

        vector<Record> vec;
        int hh=0, ll=0;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<rs)
            {
              n = mid+1;
            }
            else if (rs<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              hh = mid;
              break;
            }
         }
         if (n==m) {hh = n;}
        n = 0; m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<re)
            {
              n = mid+1;
            }
            else if (re<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              ll = mid;
              break;
            }
         }
         if (n==m) {ll = n;}

        auto yy = list.atIndex(hh);
        for (int ii=hh; ii<=ll; ii++)
        {
             vec.push_back(*yy);
             ++yy;
        }
        return vec;
    }
    void updateRecord(Key k, Value val)
    {
        int hh=0;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<k)
            {
              n = mid+1;
            }
            else if (k<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              (*(list.atIndex(mid))).v = val;
              break;
            }
         }
         if (!(k<((*(list.atIndex(n))).k)||((*(list.atIndex(n))).k)<k)) {(*(list.atIndex(n))).v
= val;}
         else
         {
            return;
         }


    }
    void updateRangeRecord(Key rs, Key re, Value val)
    {
        int hh=0, ll=0;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<rs)
            {
              n = mid+1;
            }
            else if (rs<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              hh = mid;
              break;
            }
         }
         if (n==m) {hh = n;}
        n = 0; m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<re)
            {
              n = mid+1;
            }
            else if (re<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              ll = mid;
              break;
            }
         }
         if (n==m) {ll = n;}
        auto yy = list.atIndex(hh);
        for (int ii=hh; ii<=ll; ii++)
        {
             (*yy).v = val;
             ++yy;
        }
    }
    void deleteRecord(Key k)
    {
        int hh=0;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<k)
            {
              n = mid+1;
            }
            else if (k<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              hh = mid;
              break;
            }
         }
         if (!(k<((*(list.atIndex(n))).k)||((*(list.atIndex(n))).k)<k)) {hh = n;}
        list.erase(list.atIndex(hh));
    }
    void deleteRangeRecord(Key rs, Key re)
    {
        int hh=0, ll=0;
        int n = 0, m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<rs)
            {
              n = mid+1;
            }
            else if (rs<(*(list.atIndex(mid))).k)
            {
              m = mid-1;
            }
            else{
              hh = mid;
              break;
            }
         }
         if (n==m) {hh = n;}
        n = 0; m = list.lstid();
        while(n<m)
        {
            int mid = (n+m)/2;
            if ((*(list.atIndex(mid))).k<re)
            {
```

```
            n = mid+1;
          }
          else if (re<(*(list.atIndex(mid))).k)
          {
            n = mid-1;
          }
          else{
            ll = mid;
            break;
          }
       }
       if (n==m) {ll = n;}
      auto yy = list.atIndex(hh);
      for (int ii=hh; ii<=ll; ii++)
      {
           list.erase(yy);
           --yy;
           ++yy;
      }
  }
  Record getMinRecord()
  {
      return *(list.atIndex(0));
  }
  Record getMaxRecord()
  {
      return *(list.atIndex(list.lstid()));
  }
  vector<Record> getallRecords()
  {
      vector<Record> v;
      auto yy = (list.atIndex(0));
      for (int ii=0; ii<=list.lstid(); ii++)
      {
          v.push_back(*yy);
          ++yy;
      }
      return v;
  }
};

class StudentsKey {
public:
    string rollNo;
    StudentsKey(const string &inp = "") : rollNo(inp) {}
    bool operator<(StudentsKey &rhs) { return rollNo < rhs.rollNo; }
    friend ostream & operator<<(ostream & out, StudentsKey &k);
};
ostream & operator<<(ostream & out, StudentsKey &k) {
    out << k.rollNo;
      return out;
}

class StudentsValue {
public:
    string name;
    string hostel;
    float cgpa;
    StudentsValue(const string &n = "", const string &h = "", float c = 10.0) : name(n),
hostel(h), cgpa(c) {}
    friend ostream & operator<<(ostream & out, StudentsValue &v);
};

ostream & operator<<(std::ostream & out, StudentsValue &v) {
    out << v.name << " " << v.hostel << " " << v.cgpa;
    return out;
```

```
}

class FacultyKey {
public:
    string empId;
    FacultyKey(const string &inp = "") : empId(inp) {}
    bool operator<(FacultyKey &rhs) { return empId < rhs.empId; }
    friend ostream & operator<<(ostream & out, FacultyKey &k);
};

ostream & operator<<(ostream & out, FacultyKey &k) {
    out << k.empId;
     return out;
}

class FacultyValue {
public:
    string name;
    string designation;
    float salary;
    FacultyValue(const string &n = "", const string &d = "", float s = 10.0) : name(n),
designation(d), salary(s) {}
    friend ostream & operator<<(ostream & out, FacultyValue &v);
};

ostream & operator<<(std::ostream & out, FacultyValue &v) {
    out << v.name << " " << v.designation << " " << v.salary;
    return out;
}

class CourseKey {
public:
    string courseId;
    CourseKey(const string &inp = "") : courseId(inp) {}
    bool operator<(CourseKey &rhs) { return courseId < rhs.courseId; }
    friend ostream & operator<<(ostream & out, CourseKey &k);
};

ostream & operator<<(ostream & out, CourseKey &k) {
    out << k.courseId;
    return out;
}

class CourseValue {
public:
    string name;
    string semester;
    string facultyName;
    CourseValue(const string &n = "", const string &s = "", const string &f = "") : name(n),
semester(s), facultyName(f) {}
    friend ostream & operator<<(ostream & out, FacultyValue &v);
};

ostream & operator<<(std::ostream & out, CourseValue &v) {
    out << v.name << " " << v.semester << " " << v.facultyName;
    return out;
}

int main() {
    int command;
    bool b;
    int BREAKING_COMMAND = 50;

    Database<StudentsKey, StudentsValue> student_db;
    Database<FacultyKey, FacultyValue> faculty_db;
    Database<CourseKey, CourseValue> course_db;

    while (true) {
        cin >> command;
        if (command == BREAKING_COMMAND) {
            break;
        }

        if (command == 1) { /*insert record*/
            int numberOfRecords = 0; /*number of records to be inserted*/
            cin >> numberOfRecords;

            while (numberOfRecords--) {

                string database;
                cin >> database;

                if (database == "STUDENT") {
                    string rollNo, name, hostel;
                    float cgpa;
                    cin >> rollNo >> name >> hostel >> cgpa;

                    StudentsKey k(rollNo);
                    StudentsValue v(name, hostel, cgpa);
                    student_db.insertRecord(k, v);
                } else if (database == "FACULTY") {
                    string empId, name, designation;
                    float salary;
                    cin >> empId >> name >> designation >> salary;

                    FacultyKey k(empId);
                    FacultyValue v(name, designation, salary);
                    faculty_db.insertRecord(k, v);
                } else {
                    string courseId, name, semester, facultyName;
                    cin >> courseId >> name >> semester >> facultyName;

                    CourseKey k(courseId);
                    CourseValue v(name, semester, facultyName);
                    course_db.insertRecord(k, v);
                }
            }

        } else if (command == 2) { /*check whether the key is present*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rollNo;
                cin >> rollNo;

                StudentsKey k(rollNo);
                b = student_db.isPresent(k);
                cout << b << endl;
            } else if (database == "FACULTY") {
                string empId;
                cin >> empId;

                FacultyKey k(empId);
                b = faculty_db.isPresent(k);
                cout << b << endl;
            } else {
                string courseId;
                cin >> courseId;

                CourseKey k(courseId);
                b = course_db.isPresent(k);
```

```cpp
                cout << b << endl;
            }

        } else if (command == 3) {
            string database; /*returns the record containing the key*/
            cin >> database;

            if (database == "STUDENT") {
                string rollNo;
                cin >> rollNo;

                StudentsKey k(rollNo);
                auto r = student_db.selectRecord(k);
                cout << r.getKey() << " " << r.getValue() << endl;
            } else if (database == "FACULTY") {
                string empId;
                cin >> empId;

                FacultyKey k(empId);
                auto r = faculty_db.selectRecord(k);
                cout << r.getKey() << " " << r.getValue() << endl;
            } else {
                string courseId;
                cin >> courseId;

                CourseKey k(courseId);
                auto r = course_db.selectRecord(k);
                cout << r.getKey() << " " << r.getValue() << endl;
            }

        } else if (command == 4) { /*returns a vector containing all the records in the database
                                     whose key lies between rangeStart and rangeEnd (inclusive).*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                StudentsKey rs(rangeStart);
                StudentsKey re(rangeEnd);
                auto lst = student_db.selectRangeRecord(rs, re);
                for (auto r : lst) {
                    cout << r.getKey() << " " << r.getValue() << endl;
                }
            } else if (database == "FACULTY") {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                FacultyKey rs(rangeStart);
                FacultyKey re(rangeEnd);
                auto lst = faculty_db.selectRangeRecord(rs, re);
                for (auto r : lst) {
                    cout << r.getKey() << " " << r.getValue() << endl;
                }
            } else {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                CourseKey rs(rangeStart);
                CourseKey re(rangeEnd);
                auto lst = course_db.selectRangeRecord(rs, re);
                for (auto r : lst) {
                    cout << r.getKey() << " " << r.getValue() << endl;
                }
            }

        } else if (command == 5) { /*updates the record in the database whose key is k with value v.
                                     If k is not present, there is no change in the database.*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rollNo, name, hostel;
                float cgpa;
                cin >> rollNo >> name >> hostel >> cgpa;

                StudentsKey k(rollNo);
                StudentsValue v(name, hostel, cgpa);
                student_db.updateRecord(k, v);
            } else if (database == "FACULTY") {
                string empId, name, designation;
                float salary;
                cin >> empId >> name >> designation >> salary;

                FacultyKey k(empId);
                FacultyValue v(name, designation, salary);
                faculty_db.updateRecord(k, v);
            } else {
                string courseId, name, semester, facultyName;
                cin >> courseId >> name >> semester >> facultyName;

                CourseKey k(courseId);
                CourseValue v(name, semester, facultyName);
                course_db.updateRecord(k, v);
            }

        } else if (command == 6) { /*updates all records in the database whose key lies between
                                     rangeStart and rangeEnd, with value v (inclusive).*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rangeStart, rangeEnd, name, hostel;
                float cgpa;
                cin >> rangeStart >> rangeEnd >> name >> hostel >> cgpa;

                StudentsKey rs(rangeStart);
                StudentsKey re(rangeEnd);
                StudentsValue v(name, hostel, cgpa);
                student_db.updateRangeRecord(rs, re, v);
            } else if (database == "FACULTY") {
                string rangeStart, rangeEnd, name, designation;
                float salary;
                cin >> rangeStart >> rangeEnd >> name >> designation >> salary;

                FacultyKey rs(rangeStart);
                FacultyKey re(rangeEnd);
                FacultyValue v(name, designation, salary);
                faculty_db.updateRangeRecord(rs, re, v);
            } else {
                string rangeStart, rangeEnd, name, semester, facultyName;
                cin >> rangeStart >> rangeEnd >> name >> semester >> facultyName;

                CourseKey rs(rangeStart);
                CourseKey re(rangeEnd);
                CourseValue v(name, semester, facultyName);
                course_db.updateRangeRecord(rs, re, v);
            }

        } else if (command == 7) { /*deletes the record with key k.*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rollNo;
                cin >> rollNo;

                StudentsKey k(rollNo);
                student_db.deleteRecord(k);
            } else if (database == "FACULTY") {
                string empId;
                cin >> empId;

                FacultyKey k(empId);
                faculty_db.deleteRecord(k);
            } else {
                string courseId;
                cin >> courseId;

                CourseKey k(courseId);
                course_db.deleteRecord(k);
            }

        } else if (command == 8) { /*deletes all records with key between rangeStart and rangeEnd (inclusive).*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                StudentsKey rs(rangeStart);
                StudentsKey re(rangeEnd);
                student_db.deleteRangeRecord(rs, re);
            } else if (database == "FACULTY") {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                FacultyKey rs(rangeStart);
                FacultyKey re(rangeEnd);
                faculty_db.deleteRangeRecord(rs, re);
            } else {
                string rangeStart, rangeEnd;
                cin >> rangeStart >> rangeEnd;

                CourseKey rs(rangeStart);
                CourseKey re(rangeEnd);
                course_db.deleteRangeRecord(rs, re);
            }

        } else if (command == 9) { /*return the minimum record*/
            string database;
            cin >> database;

            if (database == "STUDENT") {
                auto r = student_db.getMinRecord();
                cout << r.getKey() << " " << r.getValue() << endl;
            } else if (database == "FACULTY") {
                auto r = faculty_db.getMinRecord();
                cout << r.getKey() << " " << r.getValue() << endl;
            } else {
                auto r = course_db.getMinRecord();
                cout << r.getKey() << " " << r.getValue() << endl;
```

```
      }

    } else if (command == 10) { /*return the maximum record*/
      string database;
      cin >> database;

      if (database == "STUDENT") {
        auto r = student_db.getMaxRecord();
        cout << r.getKey() << " " << r.getValue() << endl;
      } else if (database == "FACULTY") {
        auto r = faculty_db.getMaxRecord();
        cout << r.getKey() << " " << r.getValue() << endl;
      } else {
        auto r = course_db.getMaxRecord();
        cout << r.getKey() << " " << r.getValue() << endl;
      }

    } else if (command == 11) { /*returns a vector containing all the records in the database.*/
      string database;
      cin >> database;

      if (database == "STUDENT") {
        auto lst = student_db.getallRecords();
        for (auto r : lst) {
          cout << r.getKey() << " " << r.getValue() << endl;
        }
      } else if (database == "FACULTY") {
        auto lst = faculty_db.getallRecords();
        for (auto r : lst) {
          cout << r.getKey() << " " << r.getValue() << endl;
        }
      } else {
        auto lst = course_db.getallRecords();
        for (auto r : lst) {
          cout << r.getKey() << " " << r.getValue() << endl;
        }
      }

    } else {
      cout << "INVALID COMMAND!" << endl;
      break;
    }
  }
}
```

Line: 1 Col: 1

Upload Code as File   Test against custom input   Run Code   Submit Code