#include <list>

#include<iostream>

using namespace std;

class HashTable

{

    int capacity;

  list<int> \*table;

    private:

        int \*arr;

        int \*temp;

        int \*temp2;

        // int size;

        int flag;

        int percent;

        int count;

    public:

        int size;

        HashTable(int s);

        void insertion(int num);

        void deletion(int num);

        void checking();

        int searching(int num);

        void show();

        void insertItem(int key, int data);

        void deleteItem(int key);

        // ~hashing();

        int checkPrime(int n)

  {

  int i;

  if (n == 1 || n == 0)

  {

    return 0;

  }

  for (i = 2; i < n / 2; i++)

  {

    if (n % i == 0)

    {

    return 0;

    }

  }

  return 1;

  }

  int getPrime(int n)

  {

      if (n % 2 == 0)

      {

        n++;

      }

      while (!checkPrime(n))

      {

        n += 2;

      }

  return n;

  }

  int hashFunction(int key)

  {

    return (key % capacity);

  }

  void displayHash();

};

HashTable::HashTable(int s)

{

    size = s;

    percent = s/2;

    arr = new int[size];

    temp = NULL;

    count = 0;

    flag = 0;

    int size = getPrime(s);

  this->capacity = size;

  table = new list<int>[capacity];

}

void HashTable::insertion(int n)

{

    int Rem = n % size;

    if (arr[Rem] == 0)

    {

        arr[Rem] = n;

        count += 1;

        if (flag == 0)

        {

            checking();

        }

    }

    else

    {

        while(arr[Rem] != 0)

        {

            Rem += 1;

        }

        arr[Rem] = n;

        count += 1;

        if (flag == 0)

        {

            checking();

        }

    }

}

void HashTable::insertItem(int key, int data)

{

  int index = hashFunction(key);

  table[index].push\_back(data);

}

void HashTable::checking()

{

    if (count == percent)

    {

        flag = 1;

        temp = arr;

        int newSize = searching(size\*2);

        temp2 = new int[newSize];

        int preSize = size;

        size = newSize;

        percent = size /2;

        arr = temp2;

        count = 0;

        for (int i = 0; i < preSize; i++)

        {

            if (temp[i] != 0)

            {

                insertion(temp[i]);

            }

        }

        flag = 0;

    }

}

void HashTable::deleteItem(int key)

{

  int index = hashFunction(key);

  list<int>::iterator i;

  for (i = table[index].begin(); i != table[index].end(); i++)

  {

  if (\*i == key)

    break;

  }

  if (i != table[index].end())

  table[index].erase(i);

}

void HashTable::displayHash()

{

for (int i = 0; i < capacity; i++)

  {

  cout << "table[" << i << "]";

  for (auto x : table[i])

    cout << " --> " << x;

  cout << endl;

  }

}

int HashTable::searching(int n)

{

    int flag1 = 0;

    while(1)

    {

        for (int i = 2; i < (n+1)/2; i++)

        {

            if (n % i == 0)

            {

                flag1 = 1;

                break;

            }

        }

        if (flag1 == 1)

        {

            n += 1;

            flag1 = 0;

        }

        else

            return n;

    }

}

void HashTable ::show()

{

    for (int i = 0; i < size; i++)

    {

        cout<<arr[i]<<endl;

    }

}

int main()

{

    HashTable obj(5);

    obj.insertion(13);

    obj.insertion(24);

    obj.insertion(23);

    obj.insertion(34);

    obj.insertion(50);

    obj.show();

    int key[] = {231, 321, 212, 321, 433, 262};

  int data[] = {123, 432, 523, 43, 423, 111};

  int size = sizeof(key) / sizeof(key[0]);

  //HashTable h(size);

  HashTable h(12);

  for (int i = 0; i < size; i++)

        h.insertItem(key[i], data[i]);

  h.deleteItem(231);

  h.displayHash();

    return 0;

}