Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

ОТЧЁТ

по лабораторной работе №8 (Week 8 Openedu)

Студент Луговских Савелий P3218

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Задача 1 Множество

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

Формат входного файла

В первой строке входного файла находится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:

- `A x` — добавить элемент x в множество. Если элемент уже есть в множестве, то ничего делать не надо.
- `D x` — удалить элемент x. Если элемента x нет, то ничего делать не надо.
- `? x` — если ключ x есть в множестве, выведите `Y`, если нет, то выведите `N`.

Аргументы указанных выше операций — целые числа, не превышающие по модулю 10^{18}.

Формат выходного файла

Выведите последовательно результат выполнения всех операций «`?`». Следуйте формату выходного файла из примера.

Пример

input.txt	output.txt
8 A 2 A 5 A 3 ? 2 ? 4 A 2	Y N N

D 2 ? 2	

Исходный код к задаче 1

```
using
System.Collections.Generic;

namespace Dictionary
{
    using System.IO;

    class Program
    {
        static void Main()
        {
            StreamReader input = new
StreamReader("input.txt");
            int length = int.Parse(input.ReadLine());
            Dictionary<string, bool> dictionary = new
Dictionary<string, bool>(length);
            StreamWriter output = new
StreamWriter("output.txt");
            for (int i = 1; i <= length; i++)
            {
                string[] tokens = input.ReadLine().Split('
');
                switch (tokens[0][0])
                {
                    case 'A':
                        {
                            if
(!dictionary.ContainsKey(tokens[1]))
                            {
                                dictionary[tokens[1]] =
true;
```

```
                }

                break;
            }

        case 'D':
            {
                if
(dictionary.ContainsKey(tokens[1]))
                {

dictionary.Remove(tokens[1]);
                }

                break;
            }

        case '?':
            {
                if
(dictionary.ContainsKey(tokens[1]))
                {
                    output.WriteLine('Y');
                }
                else
                {
                    output.WriteLine('N');
                }

                break;
            }
    }
```

```
                    }


                    input.Close();

                    output.Close();

                }

            }

        }
```

```
def
mergeSort(listToSort,
left, right,output):
                if len(listToSort) > 1:
                    mid = len(listToSort) // 2
                    lefthalf = listToSort[:mid]
                    righthalf = listToSort[mid:]

                    mergeSort(lefthalf, left, left + mid-1, output)
                    mergeSort(righthalf, left+mid, right,output)

                    i = 0
                    j = 0
                    k = 0

                    while i < len(lefthalf) and j < len(righthalf):
                        if lefthalf[i] < righthalf[j]:
                            listToSort[k] = lefthalf[i]
                            i += 1
                        else:
                            listToSort[k] = righthalf[j]
                            j += 1
                            k += 1

                    #первый дошли до конца
                    while i < len(lefthalf):
                        listToSort[k] = lefthalf[i]
                        i += 1
                        k += 1

                    #Второй дошли ли до конца
                    while j < len(righthalf):
                        listToSort[k] = righthalf[j]
                        j += 1
                        k += 1
                    temp_str = "%s %s %s %s\n"
            %(left,right,listToSort[0],listToSort[len(listToSort)-1])
```

```
            output.writelines(temp_str)
input = open('input.txt');
output = open('output.txt' , 'w');
arrLength = int(input.readline())
list = []
for line in input.readline().split(' '):
    list.append(int(line))
mergeSort(list, 1, len(list), output)
for x in list:
    output.write(str(x) + " ")
```

Бенчмарк к задаче 1

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.593	65134592	11189636	501237
1	OK	0.015	10067968	43	9
2	OK	0.062	10051584	8	3
3	OK	0.031	10100736	51	12
4	OK	0.031	10080256	542	99
5	OK	0.015	10170368	618	54
6	OK	0.031	10407936	5451	1038
7	OK	0.031	10403840	6436	957
8	OK	0.015	10358784	13382	957
9	OK	0.015	10514432	22394	981
10	OK	0.015	10375168	7030	465
11	OK	0.015	10350592	7020	411
12	OK	0.031	11558912	63829	10002
13	OK	0.031	12021760	80339	4947
14	OK	0.031	11997184	80203	5034
15	OK	0.046	11718656	545113	100323
16	OK	0.046	11767808	639485	99282
17	OK	0.062	11968512	738870	99558
18	OK	0.078	17387520	1338668	99636
19	OK	0.078	21184512	2237627	99540
20	OK	0.093	21712896	903052	50202
21	OK	0.093	21684224	902843	49536
22	OK	0.171	27197440	2725205	501237
23	OK	0.187	27185152	3196877	499713

Задача 2 Прошитый ассоциативный массив

Имя входного файла:	input.txt
Имя выходного файла:	output.txt

Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте прошитый ассоциативный массив.

Формат входного файла

В первой строке входного файла находится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:

- `get x` — если ключ x есть в множестве, выведите соответствующее ему значение, если нет, то выведите `<none>`.
- `prev x` — вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен позже всех, но до x, или `<none>`, если такого нет или в массиве нет x.
- `next x` — вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен раньше всех, но после x, или `<none>`, если такого нет или в массиве нет x.
- `put x y` — поставить в соответствие ключу x значение y. При этом следует учесть, что:
  - если, независимо от предыстории, этого ключа на момент вставки в массиве не было, то он считается только что вставленным и оказывается самым последним среди добавленных элементов — то есть, вызов `next` с этим же ключом сразу после выполнения текущей операции `put`должен вернуть `<none>`;
  - если этот ключ уже есть в массиве, то значение необходимо изменить, и в этом случае ключ не считается вставленным еще раз, то есть, не меняет своего положения в порядке добавленных элементов.
- `delete x` — удалить ключ x. Если ключа x в ассоциативном массиве нет, то ничего делать не надо.

Ключи и значения — строки из латинских букв длиной не менее одного и не более 20 символов.

Формат выходного файла

Выведите последовательно результат выполнения всех операций `get`, `prev`, `next`. Следуйте формату выходного файла из примера.

Пример

input.txt output.txt

14	c
put zero a	b
put one b	d
put two c	c
put three d	a
put four e	e
get two	<none>
prev two	
next two	
delete one	
delete three	
get two	
prev two	
next two	
next four	

Исходный код к задаче 2

```
namespace
Eighth_week
{
    using System.Collections.Generic;
    using System.IO;

    class Program
    {
        static void Main()
        {
            StreamReader input = new StreamReader("input.txt");
            int length = int.Parse(input.ReadLine());
            Dictionary<string, int> positionDictionary = new Dictionary<string,
int>(length);
            (string, int, int)[] valueArray = new (string, int, int)[length];
            int currentPostion = 0, lastPosition, nextPosition;
            StreamWriter output = new StreamWriter("output.txt");
            valueArray[0].Item2 = -1;
            for (int i = 1; i <= length; ++i)
```

```
            {
                string[] tokens = input.ReadLine().Split(' ');
                switch (tokens[0])
                {
                    case "get":
                        {
                            if (positionDictionary.ContainsKey(tokens[1]))
                            {
output.WriteLine(valueArray[positionDictionary[tokens[1]]].Item1);
                            }
                            else
                            {
                                output.WriteLine("<none>");
                            }

                            break;
                        }

                    case "prev":
                        {
                            if (positionDictionary.ContainsKey(tokens[1]))
                            {
                                if (valueArray[positionDictionary[tokens[1]]].Item2
>= 0)
                                {
output.WriteLine(valueArray[valueArray[positionDictionary[tokens[1]]].Item2].Item1);
                                }
                                else
                                {
                                    output.WriteLine("<none>");
```

```
                }
            }
            else
            {
                output.WriteLine("<none>");
            }

            break;
        }

    case "next":
        {
            if (positionDictionary.ContainsKey(tokens[1]))
            {
                if (valueArray[positionDictionary[tokens[1]]].Item3
< currentPostion)
                {

output.WriteLine(valueArray[valueArray[positionDictionary[tokens[1]]].Item3].Item1);
                }
                else
                {
                    output.WriteLine("<none>");
                }
            }
            else
            {
                output.WriteLine("<none>");
            }

            break;
        }
```

```
                    case "put":
                        {
                            if (positionDictionary.ContainsKey(tokens[1]))
                            {
                                valueArray[positionDictionary[tokens[1]]].Item1 =
tokens[2];
                            }
                            else
                            {
                                positionDictionary[tokens[1]] = currentPostion;
                                valueArray[currentPostion].Item1 = tokens[2];
                                valueArray[currentPostion].Item3 = currentPostion +
1;
                                ++currentPostion;
                                valueArray[currentPostion].Item2 = currentPostion -
1;
                            }

                            break;
                        }

                    case "delete":
                        {
                            if (positionDictionary.ContainsKey(tokens[1]))
                            {
                                lastPosition =
valueArray[positionDictionary[tokens[1]]].Item2;
                                nextPosition =
valueArray[positionDictionary[tokens[1]]].Item3;
                                if (lastPosition >= 0)
                                {
                                    valueArray[lastPosition].Item3 = nextPosition;
```

```
                                        }


                                        if (nextPosition <= currentPostion)
                                        {
                                            valueArray[nextPosition].Item2 = lastPosition;
                                        }


                                        positionDictionary.Remove(tokens[1]);
                                    }


                                    break;
                                }
                            }
                        }


                        input.Close();
                        output.Close();
                    }
                }
            }
namespace
Inversion
            {
                using System.Collections.Generic;
                using System;
                using System.IO;
                using System.Linq;

                class Program
                {
                    public static long inversion;

                    public static StreamWriter output = new
            StreamWriter("output.txt");

                    static List<int> Merge(List<int> list,
            int leftBoard, int rightBoard)
```

```
        {
            if (rightBoard == leftBoard)
            {
                return list.GetRange(leftBoard,
1);
            }
            else
            {
                List<int> sortArray = new
List<int>();
                int i = 0, j = 0, middleRight =
(int)Math.Ceiling((double)(rightBoard +
leftBoard) / 2);
                List<int> leftList = Merge(list,
leftBoard, middleRight - 1);
                List<int> rightList =
Merge(list, middleRight, rightBoard);
                while (i < leftList.Count || j <
rightList.Count)
                {
                    if (i == leftList.Count || j
< rightList.Count && leftList[i] > rightList[j])
                    {
                        inversion +=
leftList.Count - i;

sortArray.Add(rightList[j]);
                        j++;
                    }
                    else
                    {

sortArray.Add(leftList[i]);
                        i++;
                    }
                }

                return sortArray;
            }
        }

        static void Main(string[] args)
        {
            StreamReader input = new
StreamReader("input.txt");
            input.ReadLine();
            List<int> list = new List<int>();
```

```
                foreach (var token in
input.ReadLine().Split(' '))
                {
                    list.Add(int.Parse(token));
                }

                Merge(list, 0, list.Count - 1);
                input.Close();
                output.WriteLine(inversion);
                output.Close();
            }
        }
    }
```

Бенчмарк к задаче 2

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.171	120655872	23499808	10303658
1	OK	0.031	10489856	158	26
2	OK	0.015	10543104	12	8
3	OK	0.031	10534912	25	5
4	OK	0.031	10416128	25	8
5	OK	0.015	10444800	82	20
6	OK	0.031	10477568	1200	504
7	OK	0.031	10534912	1562	564
8	OK	0.031	10772480	12204	4617
9	OK	0.015	10809344	12058	4340
10	OK	0.078	16879616	960183	395964
11	OK	0.062	17149952	1318345	765350
12	OK	0.078	17514496	1420595	880052
13	OK	0.093	16764928	1079934	395020
14	OK	0.078	16822272	840022	332970
15	OK	0.062	17584128	1223121	889998
16	OK	0.093	24346624	3120970	486100
17	OK	0.093	23826432	3123298	486652
18	OK	0.093	23769088	3122193	479024
19	OK	0.109	16879616	900630	420456
20	OK	0.078	23851008	3121195	486718
21	OK	0.203	37236736	4199992	8
22	OK	0.187	37572608	4099993	8
23	OK	0.187	37621760	3999994	8