Министерство образования и науки Российской Федерации
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Дисциплина «**Алгоритмы и структуры данных**»

ОТЧЁТ

по лабораторным работам за 8 неделю

Студент **Пастухов К.А. группы P3218**

Санкт-Петербург
2018 г.

Множество

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

Выполнение

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Globalization;

namespace Openedu.Week8
{
    public class HashTable
    {
        private int tableSize;
        private LinkedList<long>[] table;

        public HashTable(int size = 1500)
        {
            tableSize = size;
            table = new LinkedList<long>[size];
            for (int i = 0; i < size; i++)
                table[i] = new LinkedList<long>();
        }

        public void Insert(long key)
        {
            int hash = GetHash(key);
            if (!table[hash].Contains(key))
                table[hash].AddLast(key);
        }
        public void Remove(long key)
        {
            int hash = GetHash(key);
            table[hash].Remove(key);
        }

        public bool ContainsKey(long key)
        {
            int hash = GetHash(key);
            return table[hash].Contains(key);
        }

        private int GetHash(long key)
        {
            return Math.Abs((((int)key).GetHashCode() + (key >>
32).GetHashCode()) % tableSize);
        }
    }
    public class Task1
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
```

```
                HashTable hashTable = new HashTable(n);
                for (int i = 0; i < n; i++)
                {
                    string[] str = streamReader.ReadLine().Split(' ');
                    switch(str[0][0])
                    {
                        case 'A':
                            hashTable.Insert(long.Parse(str[1]));
                            break;
                        case 'D':
                            hashTable.Remove(long.Parse(str[1]));
                            break;
                        case '?':

streamWriter.WriteLine(hashTable.ContainsKey(long.Parse(str[1])) ? "Y" : "N");
                            break;
                    }
                }
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.796	69296128	11189636	501237
1	OK	0.015	11227136	43	9
2	OK	0.031	11173888	8	3
3	OK	0.031	11194368	51	12
4	OK	0.031	11202560	542	99
5	OK	0.015	11247616	618	54
6	OK	0.046	11522048	5451	1038
7	OK	0.046	11493376	6436	957
8	OK	0.015	11579392	13382	957
9	OK	0.015	11636736	22394	981
10	OK	0.031	11554816	7030	465
11	OK	0.046	11497472	7020	411
12	OK	0.062	14405632	63829	10002
13	OK	0.015	14417920	80339	4947
14	OK	0.031	14442496	80203	5034
15	OK	0.078	26120192	545113	100323
16	OK	0.093	26083328	639485	99282
17	OK	0.109	26148864	738870	99558
18	OK	0.171	26165248	1338668	99636
19	OK	0.218	26152960	2237627	99540
20	OK	0.156	26230784	903052	50202
21	OK	0.156	26279936	902843	49536
22	OK	0.296	47640576	2725205	501237
23	OK	0.312	46522368	3196877	499713
24	OK	0.343	47681536	3694712	501051
25	OK	1.281	50851840	6694340	500355
26	OK	1.546	50409472	11189636	500040
27	OK	0.859	60280832	4902931	249012
28	OK	0.843	60194816	4902757	250305
29	OK	0.843	69091328	9687139	300000
30	OK	0.750	69296128	9687570	300000
31	OK	0.718	66142208	8000008	300000
32	OK	1.796	68804608	11000008	150000

Прошитый ассоциативный массив

Реализуйте прошитый ассоциативный массив.

Выполнение

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Globalization;

namespace Openedu.Week8
{
    public class HashTable
    {
        Dictionary<string, LinkedListNode<string>> table;
        private LinkedList<string> history;

        public HashTable()
        {
            table = new Dictionary<string, LinkedListNode<string>>();
            history = new LinkedList<string>();
        }

        public void Insert(string key, string value)
        {
            if (table.ContainsKey(key))
            {
                table[key].Value = value;
            }
            else
            {
                history.AddLast(value);
                table.Add(key, history.Last);
            }
        }
        public bool TryGetValue(string key, out string value)
        {
            if (table.TryGetValue(key, out LinkedListNode<string> node))
            {
                value = node.Value;
                return true;
            }
            value = null;
            return false;
        }
        public bool TryGetNext(string key, out string value)
        {
            if (table.TryGetValue(key, out LinkedListNode<string> node) &&
node.Next != null)
            {
                value = table[key].Next.Value;
                return true;
            }
            value = null;
            return false;
        }
        public bool TryGetPrevious(string key, out string value)
        {
            if (table.TryGetValue(key, out LinkedListNode<string> node) &&
node.Previous != null)
```

```
            {
                value = table[key].Previous.Value;
                return true;
            }
            value = null;
            return false;
        }
        public void Remove(string key)
        {
            if (table.TryGetValue(key, out LinkedListNode<string> node))
            {
                table.Remove(key);
                history.Remove(node);
            }
        }
    }
    public class Task2
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
                HashTable hashTable = new HashTable();
                for (int i = 0; i < n; i++)
                {
                    string[] str = streamReader.ReadLine().Split(' ');
                    switch(str[0])
                    {
                        case "get":
                            streamWriter.WriteLine(hashTable.TryGetValue(str[1],
out string value1) ? value1 : "<none>");
                            break;
                        case "prev":

streamWriter.WriteLine(hashTable.TryGetPrevious(str[1], out string value2) ?
value2 : "<none>");
                            break;
                        case "next":
                            streamWriter.WriteLine(hashTable.TryGetNext(str[1],
out string value3) ? value3 : "<none>");
                            break;
                        case "put":
                            hashTable.Insert(str[1], str[2]);
                            break;
                        case "delete":
                            hashTable.Remove(str[1]);
                            break;
                    }
                }
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.437	169676800	23499808	10303658
1	OK	0.031	11014144	158	26
2	OK	0.046	10911744	12	8
3	OK	0.015	10956800	25	5
4	OK	0.046	10981376	25	8
5	OK	0.062	11071488	82	20
6	OK	0.031	11010048	1200	504
7	OK	0.031	11042816	1562	564
8	OK	0.031	11366400	12204	4617
9	OK	0.031	11366400	12058	4340
10	OK	0.062	12222464	960183	395964
11	OK	0.062	12275712	1318345	765350
12	OK	0.078	12247040	1420595	880052
13	OK	0.078	11993088	1079934	395020
14	OK	0.078	12038144	840022	332970
15	OK	0.078	12083200	1223121	889998
16	OK	0.109	21942272	3120970	486100
17	OK	0.093	21929984	3123298	486652
18	OK	0.109	22016000	3122193	479024
19	OK	0.078	12070912	900630	420456
20	OK	0.109	21929984	3121195	486718
21	OK	0.234	44490752	4199992	8
22	OK	0.234	44703744	4099993	8
23	OK	0.234	44888064	3999994	8
24	OK	0.234	44851200	3899995	8
25	OK	0.218	43335680	3799996	8
26	OK	0.234	42754048	3699997	8
27	OK	0.234	42139648	3599998	8
28	OK	0.234	42074112	3499999	8
29	OK	0.218	40534016	3400000	8
30	OK	0.218	40591360	3300001	8
31	OK	0.218	12001280	5399043	1973124
32	OK	0.203	12034048	4200443	1669405
33	OK	0.250	12120064	6099290	4429770
34	OK	0.546	41791488	15598672	2589784
35	OK	0.531	41656320	15589269	2586758
36	OK	0.500	42815488	15603830	2398360