Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Мегафакультет компьютерных технологий и управления

Дисциплина
«Алгоритмы и структуры данных»

Лабораторная работа №4

Выполнил:
Студент группы P3218
Рябов Сергей Витальевич
Преподаватель:
Муромцев Дмитрий Ильич

Санкт-Петербург,
2018

1. Стек

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо "+ N", либо "−". Команда "+ N" означает добавление в стек числа N, по модулю не превышающего 10^9. Команда "−" означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10^6 элементов.

Формат входного файла

В первой строке входного файла содержится M ($1 \le M \le 10^6$) — число команд. Каждая последующая строка исходного файла содержит ровно одну команду.

Формат выходного файла

Выведите числа, которые удаляются из стека с помощью команды "−", по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.

Исходный код (C#):

```
using System;
using System.IO;
using System.Collections.Generic;
namespace ADS.Week4
{
    public class Stack<T>
    {
        private List<T> elements;
        public int Size { get  { return elements.Count; } }

        public Stack()
        {
            elements = new List<T>();
        }

        public void Push(T element)
        {
            elements.Add(element);
        }

        public T Pop()
        {
            T element = elements[elements.Count - 1];
            elements.RemoveAt(elements.Count - 1);
            return element;
        }
    }

    public class Task1
    {
        public static void Main(string[] args)
        {
            int n;
            Stack<int> stack = new Stack<int>();
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
```

```
                n = int.Parse(streamReader.ReadLine());
                for (int i = 0; i < n; i++)
                {
                    String[] str = streamReader.ReadLine().Split(' ');
                    if ("+".Equals(str[0]))
                    {
                        stack.Push(int.Parse(str[1]));
                    }
                    else
                    {
                        streamWriter.WriteLine(stack.Pop());
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.453	19820544	13389454	5693807
1	OK	0.031	10526720	33	10
2	OK	0.062	10514432	11	3
3	OK	0.031	10457088	19	6
4	OK	0.031	10444800	19	6
5	OK	0.031	10452992	19	6
6	OK	0.031	10448896	96	45
7	OK	0.031	10436608	85	56
8	OK	0.015	10448896	129	11
9	OK	0.031	10461184	131	12
10	OK	0.031	10465280	859	540
11	OK	0.015	10518528	828	573
12	OK	0.015	10522624	1340	11
13	OK	0.015	10432512	1325	12
14	OK	0.031	10735616	8292	5590
15	OK	0.031	10727424	8212	5706

2. Очередь

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «−». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «−» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 106 элементов.

Формат входного файла

В первой строке содержится M ($1 \le M \le 10^6$) — число команд. В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Выведите числа, которые удаляются из очереди с помощью команды «−», по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из очереди. Гарантируется, что извлечения из пустой очереди не производится.

Исходный код (C#):

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

namespace ADS.Week4
{
    public class Queue<T>
    {
        private class LinkedElement
        {
            public T Value { get; set; }
            public LinkedElement NextElement { get; set; }
            public LinkedElement(T value)
            {
                Value = value;
            }
        }
        private LinkedElement first;
        private LinkedElement last;

        public void Enqueue(T element)
        {
            LinkedElement newElement = new LinkedElement(element);
            if (first != null)
                first.NextElement = newElement;
            else
                last = newElement;
            first = newElement;
        }
        public T Dequeue()
        {
            T value = last.Value;
            last = last.NextElement;
            if (last == null)
                first = null;
            return value;
        }
    }
    public class Task2
    {
        public static void Main(string[] args)
        {
            int n;
            Queue<int> queue = new Queue<int>();
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                n = int.Parse(streamReader.ReadLine());
                for (int i = 0; i < n; i++)
                {
                    String[] str = streamReader.ReadLine().Split(' ');
                    if ("+".Equals(str[0]))
                    {
                        queue.Enqueue(int.Parse(str[1]));
                    }
```

```
                    else
                    {
                        streamWriter.WriteLine(queue.Dequeue());
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.906	52314112	13389454	5693807
1	OK	0.046	10174464	20	7
2	OK	0.031	10084352	11	3
3	OK	0.031	10125312	19	6
4	OK	0.015	10108928	19	6
5	OK	0.031	10166272	96	45
6	OK	0.031	10203136	85	56
7	OK	0.031	10162176	129	12
8	OK	0.031	10096640	131	12
9	OK	0.031	10149888	859	538
10	OK	0.015	10174464	828	573
11	OK	0.015	10162176	1340	12
12	OK	0.015	10162176	1325	12
13	OK	0.031	10371072	8292	5589
14	OK	0.015	10407936	8212	5706

3. Скобочная последовательность

Последовательность A, состоящую из символов из множества «(», «)», «[» и «]», назовем правильной скобочной последовательностью, если выполняется одно из следующих утверждений:

- A — пустая последовательность;
- первый символ последовательности A — это «(», и в этой последовательности существует такой символ «)», что последовательность можно представить как A=(B)C, где B и C — правильные скобочные последовательности;
- первый символ последовательности A — это «[», и в этой последовательности существует такой символ «]», что последовательность можно представить как A=[B]C, где B и C — правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

Формат входного файла

Первая строка входного файла содержит число N ($1 \le N \le 500$) - число скобочных последовательностей, которые необходимо проверить. Каждая из следующих N строк содержит скобочную последовательность длиной от 1 до 10^4 включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.

Формат выходного файла

Для каждой строки входного файла выведите в выходной файл «YES», если соответствующая последовательность является правильной скобочной последовательностью, или «NO», если не является.

Исходный код (C#):

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

namespace ADS.Week4
{
    public class Task3
    {
        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());
                for (int i = 0; i < n; i++)
                {
                    Stack<char> stack = new Stack<char>();
                    String sequence = streamReader.ReadLine();
                    bool result = true;
                    for (int j = 0; j < sequence.Length; j++)
                    {
                        switch(sequence[j])
                        {
                            case '(':
                                stack.Push('(');
                                break;
                            case '[':
                                stack.Push('[');
                                break;
                            case ']':
                                if (stack.Size == 0 || stack.Pop() != '[')
                                    result = false;
                                break;
                            case ')':
                                if (stack.Size == 0 || stack.Pop() != '(')
                                result = false;
                                break;
                        }
                    }
                    streamWriter.WriteLine((result && stack.Size == 0) ? "YES" : "NO");
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.125	12378112	5000885	2133
1	OK	0.015	10412032	31	22
2	OK	0.031	10428416	15	16
3	OK	0.031	10440704	68	66
4	OK	0.031	10432512	324	256
5	OK	0.031	10518528	1541	1032
6	OK	0.031	10530816	5880	2128
7	OK	0.031	10743808	50867	2129
8	OK	0.031	11624448	500879	2110
9	OK	0.125	12378112	5000884	2120
10	OK	0.125	12238848	5000885	2133

4. Очередь с минимумом

Реализуйте работу очереди. В дополнение к стандартным операциям очереди, необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «−», либо «?». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «−» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

Формат входного файла

В первой строке содержится M ($1 \le M \le 10^6$) — число команд. В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

Исходный код (C#):

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

namespace ADS.Week4
{
    public class PriorityQueue : Queue<int>
    {
        private LinkedList<int> minimums;
        public int Minimum { get { return minimums.Last.Value; } }

        public PriorityQueue() : base()
        {
            minimums = new LinkedList<int>();
        }
```

```
        new public void Enqueue(int element)
        {
            base.Enqueue(element);
            while (minimums.Count > 0 && minimums.First.Value > element)
            {
                minimums.RemoveFirst();
            }
            minimums.AddFirst(element);
        }
        new public int Dequeue()
        {
            int value = base.Dequeue();
            if (minimums.Last.Value == value)
                minimums.RemoveLast();
            return value;
        }
    }

    public class Task4
    {
        public static void Main(string[] args)
        {
            int n;
            PriorityQueue queue = new PriorityQueue();
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                n = int.Parse(streamReader.ReadLine());
                for (int i = 0; i < n; i++)
                {
                    String[] str = streamReader.ReadLine().Split(' ');
                    switch (str[0])
                    {
                        case "+":
                            queue.Enqueue(int.Parse(str[1]));
                            break;
                        case "-":
                            queue.Dequeue();
                            break;
                        case "?":
                            streamWriter.WriteLine(queue.Minimum);
                            break;
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.468	104452096	13389342	4002151
1	OK	0.031	10932224	29	10
2	OK	0.015	10756096	11	3
3	OK	0.046	10960896	22	6
4	OK	0.031	10821632	22	6
5	OK	0.062	10862592	36	9
6	OK	0.031	10838016	48	12
7	OK	0.046	10903552	76	35
8	OK	0.031	10825728	129	12
9	OK	0.031	10821632	67	48
10	OK	0.031	10878976	44	9
11	OK	0.015	10850304	45	9
12	OK	0.031	10809344	44	9
13	OK	0.031	10842112	45	9
14	OK	0.031	10866688	721	384
15	OK	0.031	10891264	1340	12

5. Quack

Язык Quack — забавный язык, который фигурирует в одной из задач с Internet Problem Solving Contest. В этой задаче вам требуется написать интерпретатор языка Quack.

Виртуальная машина, на которой исполняется программа на языке Quack, имеет внутри себя очередь, содержащую целые числа по модулю 65536 (то есть, числа от 0 до 65535, соответствующие беззнаковому 16-битному целому типу). Слово get в описании операций означает извлечение из очереди, put — добавление в очередь. Кроме того, у виртуальной машины есть 26 регистров, которые обозначаются буквами от 'a' до 'z'. Изначально все регистры хранят нулевое значение. В языке Quack существуют следующие команды (далее под α и β подразумеваются некие абстрактные временные переменные):

+	Сложение: get α, get β, put (α+β) mod 65536
-	Вычитание: get α, get β, put (α−β) mod 65536
*	Умножение: get α, get β, put (α·β) mod 65536
/	Целочисленное деление: get α, get β, put α div β (будем считать, что α div 0 = 0)
%	Взятие по модулю: get α, get β, put α mod β (будем считать, что α mod 0 = 0)
>[register]	Положить в регистр: get α, установить значение [register] в α

<[register]	Взять из регистра: put значение [register]
P	Напечатать: get α, вывести α в стандартный поток вывода и перевести строку
P[register]	Вывести значение регистра [register] в стандартный поток вывода и перевести строку
C	Вывести как символ: get α, вывести символ с ASCII-кодом α mod 256 в стандартный поток вывода
C[register]	Вывести регистр как символ: вывести символ с ASCII-кодом α mod 256 (где α — значение регистра [register]) в стандартный поток вывода
:[label]	Метка: эта строка программы имеет метку [label]
J[label]	Переход на строку с меткой [label]
Z[register][label]	Переход если 0: если значение регистра [register] равно нулю, выполнение программы продолжается с метки [label]
E[register1][register2][label]	Переход если равны: если значения регистров [register1] и [register2] равны, исполнение программы продолжается с метки [label]
G[register1][register2][label]	Переход если больше: если значение регистра [register1] больше, чем значение регистра [register2], исполнение программы продолжается с метки [label]
Q	Завершить работу программы. Работа также завершается, если выполнение доходит до конца программы
[number]	Просто число во входном файле — put это число

Формат входного файла

Входной файл содержит синтаксически корректную программу на языке Quack. Известно, что программа завершает работу не более чем за 10^5 шагов. Программа содержит не менее одной и не более 10^5 инструкций. Метки имеют длину от 1 до 10 и состоят из цифр и латинских букв.

Формат выходного файла

Выведите содержимое стандартного потока вывода виртуальной машины в выходной файл.

Исходный код (C#):

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

namespace ADS.Week4
{
    public class Task5
    {
        public static void Main(string[] args)
        {
```

```
        using (StreamReader streamReader = new StreamReader("input.txt"))
        using (StreamWriter streamWriter = new StreamWriter("output.txt", false, Encoding.ASCII))
        {
            Quack quack = new Quack(streamReader, streamWriter);
            quack.Run();
        }
    }
}
public class Quack
{
    private Dictionary<string, int> labels;
    private Queue<ushort> memory;
    private List<string> code;
    private ushort[] registers;
    private int cursor;

    private Dictionary<char, Action<string>> commands;

    private StreamWriter streamWriter;
    private StreamReader streamReader;

    public Quack(StreamReader streamReader, StreamWriter streamWriter)
    {
        this.streamReader = streamReader;
        this.streamWriter = streamWriter;

        labels = new Dictionary<string, int>();
        memory = new Queue<ushort>();
        code = new List<string>();
        registers = new ushort['z' - 'a' + 1];
        commands = new Dictionary<char, Action<string>> {
            ['+'] = Sum,
            ['-'] = Substract,
            ['*'] = Multiply,
            ['/'] = Divide,
            ['%'] = Mod,
            ['>'] = SetRegister,
            ['<'] = GetRegister,
            ['P'] = WriteNumber,
            ['C'] = WriteChar,
            ['J'] = Jump,
            ['Z'] = JumpIfZero,
            ['E'] = JumpIfEquals,
            ['G'] = JumpIfGreater,
            ['Q'] = Quit
        };
        cursor = 0;
    }

    public void Run()
    {
        ReadCode();
        while (cursor < code.Count)
        {
            if (commands.ContainsKey(code[cursor][0]))
                commands[code[cursor][0]](code[cursor]);
            else
                Put(code[cursor]);
            cursor++;
        }
    }
    private void ReadCode()
    {
        while (!streamReader.EndOfStream)
```

```
    {
        string line = streamReader.ReadLine();
        if (':' == line[0])
            labels.Add(line.Substring(1), code.Count);
        else
            code.Add(line);
    }
}

private void Sum(string command)
{
    ushort a = memory.Dequeue();
    ushort b = memory.Dequeue();
    memory.Enqueue((ushort)(a + b));
}
private void Substract(string command)
{
    ushort a = memory.Dequeue();
    ushort b = memory.Dequeue();
    memory.Enqueue((ushort)(a - b));
}
private void Multiply(string command)
{
    ushort a = memory.Dequeue();
    ushort b = memory.Dequeue();
    memory.Enqueue((ushort)(a * b));
}
private void Divide(string command)
{
    ushort a = memory.Dequeue();
    ushort b = memory.Dequeue();
    memory.Enqueue((ushort)(a / b));
}
private void Mod(string command)
{
    ushort a = memory.Dequeue();
    ushort b = memory.Dequeue();
    memory.Enqueue((ushort)(a % b));
}
private void SetRegister(string command)
{
    registers[command[1] - 'a'] = memory.Dequeue();
}
private void GetRegister(string command)
{
    memory.Enqueue(registers[command[1] - 'a']);
}
private void WriteNumber(string command)
{
    if (command.Length == 1)
        streamWriter.WriteLine(memory.Dequeue());
    else
        streamWriter.WriteLine(registers[command[1] - 'a']);
}
private void WriteChar(string command)
{
    if (command.Length == 1)
        streamWriter.Write((char)(memory.Dequeue() % 256));
    else
        streamWriter.Write((char)(registers[command[1] - 'a'] % 256));
}
private void Jump(string command)
{
    cursor = labels[command.Substring(1)] - 1;
```

```
        }
        private void JumpIfZero(string command)
        {
            if (registers[command[1] - 'a'] == 0)
                cursor = labels[command.Substring(2)] - 1;
        }
        private void JumpIfEquals(string command)
        {
            if (registers[command[1] - 'a'] == registers[command[2] - 'a'])
                cursor = labels[command.Substring(3)] - 1;
        }
        private void JumpIfGreater(string command)
        {
            if (registers[command[1] - 'a'] > registers[command[2] - 'a'])
                cursor = labels[command.Substring(3)] - 1;
        }
        private void Quit(string command)
        {
            cursor = code.Count;
        }
        private void Put(string command)
        {
            memory.Enqueue(ushort.Parse(command));
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.109	25776128	1349803	250850
1	OK	0.031	11247616	69	6
2	OK	0.031	11321344	232	218
3	OK	0.046	11141120	3	0
4	OK	0.031	11239424	100	19
5	OK	0.046	12152832	56	58890
6	OK	0.046	11689984	67	30000
7	OK	0.046	11640832	67	30000
8	OK	0.046	11669504	55	30000
9	OK	0.031	11190272	461	60
10	OK	0.031	11386880	11235	21
11	OK	0.031	11620352	23748	42
12	OK	0.031	12201984	66906	8905
13	OK	0.031	11362304	7332	954
14	OK	0.046	11354112	4611	602
15	OK	0.031	11808768	37968	5424
16	OK	0.031	11214848	14	2
17	OK	0.031	11231232	70	10
18	OK	0.015	11235328	350	50
19	OK	0.015	11264000	1750	250
20	OK	0.031	11337728	8750	1250