Тестовое задание от Studio Mobile

Для каждой задачи решение должно быть представлено в виде отдельного консольного приложения, принимающего входные данные в указанном виде и возвращающего результат в консоль. Правильность и точность входных данных гарантируются, поэтому добавлять проверки на их корректность не нужно.

Просим вас обратить особое внимание на требования к высылаемым решениям. Кандидатам, которые соблюли их в меньшей степени, а также тем, кто решил не все задачи, присваивается более низкий приоритет.

Задача 1. Хэш-таблица

Реализовать хэш-таблицу с функцией хэширования `X % N`, где `X` — целое число, помещаемое в хэш-таблицу, `N` — целое число, от деления на которое берётся остаток. Коллизии разрешаются с помощью односвязного списка.

Интерфейс хэш-таблицы:

```
class HashTable {
    public Array<List<int>> values;
    public void Insert(int newValue);
}
```

Интерфейс элемента списка:

```
class ListNode {
    int value;
    List next;

    public void Insert(int newValue);
}
```

Реализованная программа должна выполнять добавление в хэш-таблицу элементов, заданных во входной последовательности. После ввода всех элементов программа выводит на экран содержимое хэш-таблицы.

Использование реализаций списка из стандартной библиотеки (`list`, `ArrayList`, `LinkedList`, `Hashtable` и т. п.) не допускается. В ходе решения должна быть получена своя реализация хэш-таблицы и односвязного списка с указанными интерфейсами.

Ввод

```
N
x1 x2 ... xn
```

где `N` находится в диапазоне [1; 100] — число, по делению на которое ищется остаток; `xi` находится в диапазоне $[0; 2^{32}]$ — новый элемент, добавляемый в хэш-таблицу, `n` находится в диапазоне [0; 10000].

Вывод

```
m1: xk1
m2: xk2
...
mN: xkN
```

где `mi` — остаток от деления `xi` на `N`; `xki` — число, которое имеет остаток от деления, равный `mi`.

Примеры тестовых данных

Ввод

```
5
5 100 10 24 13
```

Вывод

```
0: 5 100 10
1:
2:
3: 13
4: 4
```

Примечание: при разрешении коллизий новые элементы добавляются в конец списка. В примере выше значение 10 добавлено после значения 100, поэтому находится после него.

Задача 2. Диаграмма частоты слов

Написать генератор диаграммы относительной частоты встречаемости слов в тексте. Диаграмма должна быть представлена в текстовой форме и иметь следующий формат:

```
<слово1> <точки, отображающее относительное количество
вхождения слова1 в текст>
<слово2> <точки, отображающее относительное количество
вхождения слова2 в текст>
...
<словоN> <точки, отображающее относительное количество
вхождения словаN в текст>
```

Столбец слов и точек должен быть разделён одним пробелом.

Столбец слов должен иметь ширину, равную длине самого длинного слова, при этом более короткие слова выравниваются по правому краю (то есть, при выводе перед ними записывается нужное количество знаков подчёркивания (`_`)).

Столбец точек имеет максимальную ширину в 10 символов. 10 точек соответствуют частоте встречаемости самого частого слова. Частота встречаемости остальных слов считается относительно этого значения. Округление производится по правилам математики, при этом значения вида

ЦЕЛОЕ + 0.5 округляются вверх.

Список слов должен быть упорядочен от самого редкого до самого частого.

В тексте гарантированно могут быть только следующие символы: маленькие буквы от a до z, пробел.

Ввод

```
w1 w2 ... wn
```

где `wi` — слово, `n` находится в диапазоне [0; 10000].

Вывод

```
w1: p1
w2: p2
...
wn: pn
```

где `wi` слово из текста, `pi` — относительная частота использования, отображаемая с помощью точек.

Примеры тестовых данных

Пример 1

Ввод

```
aa aa bbb bbb bbb bbb bbb c c c c c c c c c
```

Вывод

```
_aa ..
bbb ......
__c ..........
```

Пример 2

Ввод

```
aa c c c c c c c c c c c c c c c c c c c c c c c c c c
```

Вывод

```
aa
_c ..........
```

Пример 3

Ввод

```
aa aa aa aa aa bbb bbb bbb bbb c c c c c c c c c c c c c c
```

Вывод

```
bbb ...
_aa ...
__c ..........
```

Задача 3. Простейший калькулятор

Написать программу, выполняющую операции над числами. Программа

принимает в себя арифметическое выражение в постфиксной нотации: последовательность чисел и операторов действий над двумя последними числами.

Например, входная последовательность `5 10 +` означает: `5 + 10`.

`5 10 + 10 *` означает `(5 + 10) * 10`.

`5 10 15 + -` означает `5 - (10 + 15)`.

Числа во входной последовательности гарантированно целые. Допустимые операторы: `+` (сложение), `-` (вычитание), `*` (умножение), `/` (деление).

Операторы и числа во входной последовательности отделены друг от друга пробелом.

Ввод

```
t1 t2 ... tn
```

где `ti` — токен (число или оператор); `n` находится в диапазоне [0; 10000].

Вывод

Результат вычислений.

Примеры тестовых данных

Ввод

```
5 10 + 10 * 14 -
```

Вывод

Требования к решениям

Мы настоятельно просим вас изучить требования к оформлению ваших решений. Их соблюдение ускоряет обработку результатов, а также показывает вашу готовность работать в соответствии с техническими заданиями от реальных заказчиков.

Общие требования

1. Решения должны быть выполнены на одном из следующих языков: C, C++, C#, Java, Objective-C, Swift. **Не допускается** решение на других языках, в т. ч. диалектах перечисленных языков (Kotlin, Scala и т. п.)
2. Решения должны быть высланы в виде **исходных кодов**. Нет необходимости присылать файлы проектов (.IDEA, SLN, XCODEPROJ и т. п.), скомпилированные программы или части программ (EXE, JAR, CLASS и т. п.).

   Ниже приведены допустимые расширения файлов с исходными кодами:

Язык	**Расширения файлов с исходными кодами**
C	H, C
C++	H, HPP, CPP
C#	CS
Java	JAVA
Objective-C	H, M, MM
Swift	SWIFT

3. Исходные коды должны быть рассортированы по папкам в

соответствии с номерами задач:

   - task1 — для задачи 1;
   - task2 — для задачи 2;
   - task3 — для задачи 3.

   Просьба соблюсти регистр названия папки. Исходный код решения задачи должен быть расположен в корне соответствующей папки, а не во вложенных директориях.

4. Компилируемый код должен быть кроссплатформенным и не зависеть от сторонних библиотек (т. е. должен использовать функциональность, предоставляемую стандартной библиотекой языка).

5. Приложение не должно выводить сообщений вида: «Введите данные», «Результат». Приложение просто считывает данные, решает задачу, выводит результат в указанном формате и заканчивает свою работу.

6. Не допускается внесение входных данных прямо в код приложения («хардкод»), с помощью аргументов вызова (ARGV) или из файла. Единственным источником данных для приложения является ввод пользователя.

7. Не должны применяться функции, вызывающие ожидание нажатия клавиши пользователем (`Console.ReadKey()` , `getch()` , `System.in.read()` , `cin.ignore()`). Данные в приложение приходят в указанном формате построчно; разделение входных данных в одной строке осуществляется с помощью пробела.

   В «Приложении» указан пример получения программой входных данных из консоли. Можно использовать этот код в качестве основы вашего приложения.

8. Программа должна рассчитывать только на указанный в задаче формат данных. Не допускается менять порядок и формат входных данных.

Требования к решениям на языках C и C++

Допускается использование только стандартной библиотеки. Использование сторонних библиотек (boost, curses, ncurses и т. п.) или фреймворков (Borland/Embarcadero VCL/CLX, Visual C++, Qt и т. п.) не допускается.

Сборка приложений при проверке выполняется с помощью команды:

```
g++ -std=c++14 *.cpp -O0 -o task
```

Требования к решениям на языке C#

Сборка приложений при проверке выполняется с помощью команды:

```
mcs *.cs -out:task
```

Требования к решениям на языке Java

Класс, содержащий метод `main`, должен называться `Solver`.

Файлы исходных кодов классов, относящихся к решению конкретной задачи, должны быть помещены в соответствующий `package`: `package task1`, `package task2` или `package task3`.

Просьба соблюдать регистр указанных названий.

Сборка приложений при проверке выполняется с помощью команды:

```
javac task/Solver.java
```

Требования к решениям на языках Objective-C и Swift

Допускается использование только фреймворка Foundation. Использование сторонних библиотек и фреймворков (из CocoaPods, Carthage и т. п.) не допускается.

Сборка приложений на Objective-C при проверке выполняется с помощью команды:

```
clang -fobjc-arc -fmodules *.m -o task
```

Примечание: Директива `-fobjc-arc` может быть отключена, если в решении используется MRC.

Сборка приложений на Swift при проверке выполняется с помощью команды:

```
swiftc *.swift -o task
```

Приложение

Примеры решения задачи на сложение двух чисел.

C++

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;

    cin >> a;
    cin >> b;

    cout << a + b << endl;

    return 0;
}
```

Java

Обратите внимание: класс с решением задачи называется `Solver`.

```
import java.io.*;
```

```java
import java.util.*;

class Solver
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        PrintWriter out = new PrintWriter(new
OutputStreamWriter(System.out));

        StringTokenizer strTokenizer;

        strTokenizer = new StringTokenizer(in.readLine());
        int a = Integer.parseInt(strTokenizer.nextToken());

        strTokenizer = new StringTokenizer(in.readLine());
        int b = Integer.parseInt(strTokenizer.nextToken());

        out.println(a + b);
        out.flush();
    }
}
```

C#

```csharp
using System;

class Solver
{
    public static void Main(string[] args)
    {
        string[] tokens;

        tokens = Console.ReadLine().Split(' ');
        int a = Convert.ToInt32(tokens[0]);

        tokens = Console.ReadLine().Split(' ');
        int b = Convert.ToInt32(tokens[0]);

        Console.WriteLine(a + b);
```

```
        }
    }
```