# Лаб: Списъци

Тествайте решението в Judge: <https://judge.softuni.bg/Contests/3171/Additional-Exercises>.

## Еднакви числа

Напишете програма, която да събира всички съседни числа с еднаква сума в списъкът отляво надясно.

* След като **сумирате** **двете числа** , **резултатът** може да бъде равен на **съдеда** на второто число и трябва да се **сумира също** (както е показоно по-долу).
* Винаги сумирайте **най-левите два равни съседа** (ако има няколко равни двойки съседи)

Програмата свършва, когато няма повече съседни числа с еднакви стойности. Накрая отпечатайте крайният резултат.

### Примерен вход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 3 3 6 1 | 12 1 | **3 3** 6 1 à **6 6** 1 à 12 1 |
| 8 2 2 4 8 16 | 16 8 16 | 8 **2 2** 4 8 16 à 8 **4 4** 8 16 à **8 8** 8 16 à 16 8 16 |
| 5 4 2 1 1 4 | 5 8 4 | 5 4 2 **1 1** 4 à 5 4 **2 2** 4 à 5 **4 4** 4 à 5 8 4 |

### Подсказки

Първо четем списък от числа.

Text

Description automatically generated with medium confidence

Минаваме през всички елементи. Проверяваме дали числото със сегашният индекс и следващото са еднакви. Ако е така, нулираме for-цикъла и започваме отначало. В противен случей не правим нищо.

Text

Description automatically generated

Накрая отпечатваме всички числа **разделени с интервал**.



## Трикът на Гаус

Напишете програма, която **сумира** всички **числа** в **списък** в следният формат:

*първо + последно, първо + 1 + последно - 1, първо + 2 + последно - 2, … първо + n, последно - n.*

Graphical user interface, text

Description automatically generated with medium confidence

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 1 2 3 4 5 | 6 6 3 |
| 1 2 3 4 | 5 5 |

## Обединяване на списъци

Напишете програма, която получава два списъка с числа. Крайният списък трябва да съдържа числа от двата списъка. **Първият елемент** на крайният списък е **от първия елемент на първия списък** и **вторият елемент** **е първият елемент от вторият списък** и така нататък. Ако дължината на двата списъка е **различна**, просто **добавете оставащите елеменити** в **края** на списъка.

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| **3 5 2 43 12 3 54 10 23**  **76 5 34 2 4 12** | **3** **76** **5** **5** **2** **34** **43** **2** **12** **4** **3** **12** **54** **10** **23** |
| **76 5 34 2 4 12**  **3 5 2 43 12 3 54 10 23** | **76** **3** 5 **5** **34** **2** **2** **43** **4** **12** **12** **3** **54 10 23** |

### Подсказки

* Четем два списъка
* Създаваме краен списък
* Стартираме цикъл, който минава през всички елементи и спира до дължината на по-малкия списък
* Накрая добавяме оставащите елементи (ако има някакви) на края на списък

## Списък от продукти

Прочетете числото **n** и **n** редове от продукти. Принтирайте **номериран списък** от продуктите **подредени по азбучен ред**.

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  Potatoes  Tomatoes  Onions  Apples | 1.Apples  2.Onions  3.Potatoes  4.Tomatoes |

### Подсказки

Първи четем числото **n** от конзолата.

Text

Description automatically generated

След това създаваме **списък от низове**, защото **продуктите са от низове.**



Трябва да минем **n пъти** и да **прочетем нашите продукти.**

Graphical user interface, text, application

Description automatically generated

Добавяме **продукта** към списъка.

Text

Description automatically generated

След като сме прочели продуктите, ние ги **сортираме по азбучен ред**.

Graphical user interface, text, application

Description automatically generated

* **Сортиращият метод** ги подрежда по азбучен ред.

Накрая ние **принтираме списъка**. За да направим това **използваме for-цикъл**.

A picture containing text

Description automatically generated

* Използваме **i + 1**, защото ние започваме **да броим от 1**, добавяме и **'.'** и накрая слагаме **продукта**.

## Премахнете негативните и го обърнете

Прочетете **списък от числа**. **Премахнете всички негативни** числа и принтирайте останалите числа в **обратен ред**. В случай, в които няма числа отпечатайте "empty".

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 10 -5 7 9 -33 50 | 50 9 7 10 |
| 7 -2 -10 1 | 1 7 |
| -1 -2 -3 | empty |

### Подсказки

Първо прочитаме списък от числа.

Text

Description automatically generated

Пренахваме всички негативни числа.



Ако дължината на списъка е 0 отпечатваме "empty", в противен случай принтираме всички числа разделени с интервал.

Graphical user interface, text

Description automatically generated

## Маниполация на списък

Напишете програма, която чете списък от числа. Докато не получите **"end",** ще получавате следните възможни **команди**:

**Add {число}:** Добавяне на число в края на списъка.  
**Remove {число}:** Премахване на число от списъка.  
**RemoveAt {индекс}:** Премахване на число с определен индекс.  
**Insert {число} {индекс}:** Вмъкване на число на определен индекс.

**Бележка: Не всичко ще бъде валидно!**

Когато получите **"end",** принтирате **финалния** списъка (**числата трябва да бъдат разделени с интервал**).

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4 19 2 53 6 43  Add 3  Remove 2  RemoveAt 1  Insert 8 3  end | 4 53 6 8 43 3 |

### Подсказки

Първо прочитаме списъка от конзолата.

Text

Description automatically generated

* **Разделяме** низа получен на първия ред от конзолата. След това **минаваме през всеки елемент** и го парсваме към **число**.
* Това връща **IEnumarable<int>** (**колекция** от числа) и го запазваме под формата на списък.

След входа използваме while-цикъл и switch case за различните команди.

Graphical user interface, text, application

Description automatically generated

* Спираме цикъла, когато реда е “end”, в противен случай **разделяме** входа на низвое в **tokens**.

A picture containing table

Description automatically generated

Имплементираме всички **команди**.

Text

Description automatically generated

* За всички команди с **изключение** на **"Insert"** , **tokens[1]** е винаги **число/индекс**.
* За **"Insert"** командата ние получаваме **число и индекс** (**tokens[1], tokens[2]**).

Накрая отпечатваме числата, разделени с интервал.



## List Manipulation Advanced

В тази задача ще трябва да имплентираме по-сложни команди на списъка(**разширяване на предходната задача**). Отново ще четете списък и команди, докато не получите **"end":**

**Contains {число}** – Проверявате дали число съществува в списъка - Ако е така принтирайте **"Yes**", в противен случай **"No such number ".**

**PrintEven** – принтирайте **всички четени числа разделени с интервал.**

**PrintOdd** – принтирайте **всички нечетени числа разделени с интервал.**  
**GetSum** – принтирайте **сумата на всички числа**.  
**Filter {condition} {number}** – отпечатайте всички числа, които отговарят на **даденото условие**. Условията могат да бъдат **'<', '>', ">=", "<="**.

След командата **"end"** принтирайте списъка, ако има някакви промени от оригиналния списък. **Промените** се правят само от командите от **предходната задача**.

### Примерен вход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 2 13 43 876 342 23 543  Contains 100  Contains 543  PrintEven  PrintOdd  GetSum  Filter >= 43  Filter < 100  end | No such number  Yes  2 876 342  13 43 23 543  1842  43 876 342 543  2 13 43 23 |