Упражнения: Итератори

1. Библиотека

Създайте клас Book, който трябва да има три публични свойства:

- string Title
- int Year
- List<string> Authors

Авторите може да са анонимни, един или повече. Book трябва да има само един конструктор.

Създайте клас Library, който да съхранява колекция книги и да имплементира интерфейса IEnumerable<Book> и да съдържа в себе си списък с книги:

- List<Book> books

Library може да е инициализиран без книги или с какъвто и да е брой книги и трябва да има само един конструктор.

Примери

BookLibraryExample.cs

```
 public static void Main()
 {
     Book bookOne = new Book("Animal Farm", 2003, "George Orwell");
     Book bookTwo = new Book("The Documents in the Case", 2002, "Dorothy
Sayers", "Robert Eustace");
     Book bookThree = new Book("The Documents in the Case", 1930);

     Library libraryOne = new Library();
     Library libraryTwo = new Library(bookOne, bookTwo, bookThree);
 }
```

Решение

```
public class Book
{
    public Book(string title, int year, params string[] authors)
    {
        this.Title = title;
        this.Year = year;
        this.Authors = authors;
    }

    public string Title { get; set; }

    public int Year { get; set; }

    public IReadOnlyList<string> Authors { get; set; }
}
```

```
public class Library
{
    private List<Book> books;

    public Library(params Book[] books)
    {
        this.books = new List<Book>(books);
    }
}
```

2. Итератор за библиотека

Разширете решението на предишната задача. В класа Library създайте вложен клас LibraryIterator, който трябва да имплементира интерфейса IEnumerable<Book>. Опитайте да имплементирате тялото на двата наследени метода сами. Ще ви трябват още два члена:

- List<Book> books
- int currentIndex

Сега ще можете да обхождате Library в Main() метода.

Примери

BookLibraryExample.cs

```
 public static void Main()
 {
     Book bookOne = new Book("Animal Farm", 2003, "George Orwell");
     Book bookTwo = new Book("The Documents in the Case", 2002, "Dorothy
Sayers", "Robert Eustace");
     Book bookThree = new Book("The Documents in the Case", 1930);

     Library libraryOne = new Library();
     Library libraryTwo = new Library(bookOne, bookTwo, bookThree);

     foreach (var book in libraryFull)
     {
         Console.WriteLine(book.Title);
     }
 }
```

Изход

```
Animal Farm
The Documents in the Case
The Documents in the Case
```

Решение

```
    public IEnumerator<Book> GetEnumerator()
    {
        return new LibraryIterator(this.books);
    }
    IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

    private class LibraryIterator : IEnumerator<Book>
    {
        private readonly List<Book> books;
        private int currentIndex;

        public LibraryIterator(IEnumerable<Book> books)
        {
            this.Reset();
            this.books = new List<Book>(books);
        }

        public void Dispose(){}
        public bool MoveNext() => ++this.currentIndex < this.books.Count;
        public void Reset() => this.currentIndex = -1;
        public Book Current => this.books[this.currentIndex];
        object IEnumerator.Current => this.Current;
    }
}
```

3. ListyIterator

Създайте шаблонен клас **ListyIterator**, той трябва да получава чрез своя конструктор колекцията, която ще обхожда. Трябва да съхранява елементите в списък. Класът трябва да има три основни функции:

- **Move** – да мести позицията на вътрешния индекс към следващия индекс в списъка; методът трябва да връща true ако успешно я е преместил и false ако няма следващ индекс.
- **HasNext** – да връща стойност true ако има следващ индекс и стойност false ако индексът е вече на последния елемент от списъка.
- **Print** – трябва да отпечатва елемента на дадения вътрешен индекс; изпълнението на Print върху колекция без елементи трябва да хвърля подходящо изключение със съобщението "Invalid Operation!".

По подразбиране вътрешният индекс трябва да сочи към нулевия индекс в списъка. Програмата ви трябва да поддържа следните команди:

Команда	Връщан тип	Описание
Create {e1 e2 …}	void	Създава ListyIterator от указаната колекция. В случай на команда Create без подадени елементи да създава ListyIterator с празна колекция.

Move	boolean	Тази команда трябва да мести вътрешния индекс към следваща позиция.
Print	void	Тази команда трябва да отпечатва елемента на дадения вътрешен индекс.
HasNext	boolean	Връща дали колекцията има следващ елемент.
END	void	Спира входа.

Вход

Входът ще идва от конзолата като няколко реда с команди. Първият ред винаги ще бъде единствената команда Create във входа. Последната получена команда винаги ще бъде единствената команда END.

Изход

За всяка команда от входа (с изключение на командите END и Create) отпечатайте нейният резултат на конзолата, всяка команда на нов ред. В случай на командата Move или HasNext да се отпечата върнатата стойност на методите; в случай на командата Print не е необходимо да правите нищо допълнително, тъй като самият метод трябва да се отпечатва на конзолата. Програмата трябва да прихваща всички изключения, хвърлени заради валидация (например извикването на Print при празна колекция), и да отпечатва съобщенията им.

Ограничения

- Винаги ще има само 1 Create команда и тя винаги ще е първата подадена.
- Номерата на получените команди ще са между [1...100].
- Последната команда винаги ще е единствената команда END.

Примери

Вход	Изход
Create Print END	Invalid Operation!
Create Stefcho Goshky HasNext Print Move Print END	True Stefcho True Goshky
Create 1 2 3 HasNext Move HasNext HasNext Move HasNext END	True True True True True False

4. Колекция

Като използвате **ListyIterator**-а от предишната задача, разширете го чрез имплементиране на интерфейса IEnumerable<T>; имплементирайте ръчно всички методи, изисквани от интерфейса (използвайте yield return за метода GetEnumerator()). Добавете нова команда PrintAll, която трябва да обхожда с foreach колекцията и да отпечатва всички елементи на един ред и отделени с интервал.

Вход

Входът ще идва от конзолата като няколко реда с команди. Първият ред винаги ще бъде единствената команда Create във входа. Последната получена команда винаги ще бъде единствената команда END.

Изход

За всяка команда от входа (с изключение на командите END и Create) отпечатайте резултата от тази команда на конзолата, всяка команда на нов ред. В случай на командата Move или HasNext да се отпечата върнатата стойност на методите; в случай на командата Print не е необходимо да правите нищо допълнително, тъй като самият метод трябва да се отпечатва на конзолата. В случай на командата PrintAll трябва да отпечатате всички елементи на един ред, отделени един от друг с пауза. Програмата трябва да улавя всички изключения хвърлени заради валидиране и да отпечатва съобщенията им.

Ограничения

НЕ използвайте метода GetEnumerator() от базовия клас. Използвайте своя собствена имплементация, която да ползва "yield return"

Винаги ще има само 1 Create команда и тя винаги ще е първата подадена.

Номерата на получените команди ще са между [1...100].

Последната команда винаги ще е единствената команда END.

Примери

Вход	Изход
Create 1 2 3 4 5 Move PrintAll END	True 1 2 3 4 5
Create Stefcho Goshky Peshu PrintAll Move Move Print HasNext END	Stefcho Goshky Peshu True True Peshu False

5. Стек

След като вече сте преминали курса с основни алгоритми, сега имате задачата да създадете своя версия на стек. Познавате структурата на стековете. Има колекция, която да съхранява елементите, и две функции (но не от функционалното програмиране) - за добавяне на елемент (push) и за извличане (pop). Имайте в предвид, че първият елемент който се извлича е последният в колекцията. Push метода добавя елемент в края на колекцията, а pop метода връща най-горния елемент и го премахва.

Напишете своя лична имплементация на Stack<T> и реализирайте интерфейса IEnumerable<T>. Вашата имплементация на метода GetEnumerator() трябва да спазва правилата на абстрактния тип данни Stack (т.е. да връща елементите в обратния ред на този, по който са добавени в стека).

Вход

Входът ще идва от конзолата като няколко реда с команди. Командите ще са само push и pop, следвани от цели числа за командата push и никакъв друг вход за командата pop.

Формат:

- Push {element1}, {element2}, … {elementN} – добавя дадените елементи към стека
- Pop – премахва последния вмъкнат елемент от стека

Изход

Когато получите END, с въвеждането се спира. Обходете два пъти стека с foreach и отпечатайте всички елементи, всеки на нов ред.

Ограничения

Елементите в командата push ще са валидни цели числа между [2-32…232-1].

Командите винаги ще са валидни (винаги ще са Push, Pop или END).

Ако някоя команда Pop не може да се изпълни както се очаква (например ако няма елементи в стека), отпечатайте на конзолата: "No elements".

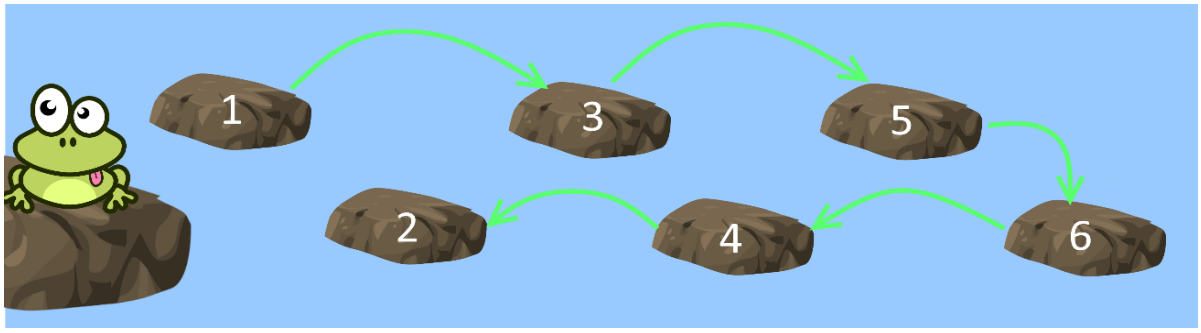
Примери

Вход	Изход
Push 1, 2, 3, 4 Pop Pop END	2 1 2 1
Push 1, 2, 3, 4 Pop Push 1 END	1 3 2 1 1 3 2 1
Push 1, 2, 3, 4 Pop Pop Pop Pop Pop	No elements

END	

6. Жабче

Да играем на една игра. Имате една мъничка Жаба и Езеро с пътечка от камъни. Всеки камък има номер. Нашата жаба трябва да пресече езерото по пътеката и после да се върне. Но има някои правила за скачане по камъните. Първо, жабата трябва да скочи на всички четни позиции върху камъните във възходящ ред, а после на всички нечетни позиции, но в низходящ ред. Редът на камъните и техните номера ще бъдат зададени на първия ред на входа. След това трябва да отпечатате реда на камъните, по който нашата жаба е скочила от единия на другия.



Опитайте да постигнете тази функционалност чрез създаване на клас Lake (той ще съдържа номерата на всички камъни подред), който имплементира интерфейса IEnumerable<int> и предефинира неговите GetEnumerator() методи.

Примери

Вход	Изход
1, 2, 3, 4, 5, 6, 7, 8	1, 3, 5, 7, 8, 6, 4, 2
1, 2, 3, 4, 5	1, 3, 5, 4, 2
13, 23, 1, -8, 4, 9	13, 1, 4, 9, -8, 23

7. *Клиника за домашни любимци

Вие сте млад и амбициозен собственик на Pet Clinics Holding. Искате от подчинените си да създадат програма, която да съхранява цялата информация за животните в базата данни. Всеки любимец трябва да има име, възраст и вид.

Вашето приложение трябва да поддържа няколко основни операции като например създаване на любимец, създаване на клиника, добавяне на любимец в клиника, изписване на любимец от клиника, отпечатване на информация за конкретна стая в клиниката и отпечатване на информация за всички стаи в клиниката.

Клиниките трябва да имат нечетен брой стаи; опитът за създаване на клиника с четен брой стаи трябва да се провали и да хвърли подходящо изключение.

Ред на настаняване

Да вземем например клиника с 5 стаи. Първата стая, в която ще се лекува любимец, е централната (стая 3). Така че редът, по който се приемат животни, е: първият любимец отива в централната (3) стая, после следващите влизат първо в лявата (2) и после в дясната (4) стая. Последните стаи, в които влизат животни, са стаи 1 и 5. В случай, че стая вече е заета я прескачаме и отиваме в

следващата в горепосочения ред. Вашата задача е да моделирате приложението и да се погрижите да поддържа някои команди.

- Първият любимец влиза в стая 3. -> 1 2 3 4 5
- След това вторият влиза в стая 2. -> 1 2 3 4 5
- Третият любимец ще влезе в стая 4. -> 1 2 3 4 5
- А последните два ще отидат в стаи 1 и 5. -> 1 2 3 4 5

След като сме покрили добавянето на любимци, време е да намерим начин да ги изписваме. Процесът на изписване не е толкова прост; когато методът за изписване е повикан, започваме от централната стая (3) и продължаваме надясно (4, 5... и така нататък) докато не открием любимец или не достигнем последната стая. Ако достигнем последната стая, започваме от първата (1) и отново се движим надясно докато не достигнем централната стая (3). Ако е открит любимец го премахваме от колекцията, спираме по-нататъшно търсене и връщаме true; ако НЕ е открит любимец, операцията връща false.

Когато е повикана команда print за стая, ако стаята съдържа любимец го отпечатайте на един ред във формата "<име на любимеца> <възраст на любимеца> <вид на любимеца>". Ако пък стаята е празна, отпечатайте "Room empty". Когато е повикана команда print за клиника, трябва да отпечатате всички стаи в клиниката по ред на техните номера.

Команди

Команда	Връщан тип	Описание
`Create Pet {име} {възраст} {вид}`	`boolean`	`Създава любимец с указаните име и възраст. (true ако операцията е успешна и false ако не е)`
`Create Clinic {име} {стаи}`	`void`	`Създава клиника с указаните име и брой стаи (ако стаите не са нечетен брой, хвърля изключение)`
`Add {име на любимец} {име на клиника}`	`boolean`	`Тази команда трябва да добави дадения любимец в указаната клиника (true ако операцията е успешна и false ако не е).`
`Release {име на клиника}`	`boolean`	`Тази команда трябва да изпише любимец от указаната клиника (true ако операцията е успешна и false ако не е).`
`HasEmptyRooms {име на клиника}`	`boolean`	`Връща дали клиниката има празни стаи. (true ако има и false ако няма).`
`Print {име на клиника}`	`void`	`Тази команда трябва да отпечата всички стаи в указаната клиника, подредени по номера.`
`Print {име на клиника} {стая}`	`void`	`Отпечатва на един ред съдържанието на конкретната стая.`

Вход

На първия ред ще ви бъде дадено цяло число N – броят команди, които ще получите. На всеки от следващите N на брой редове ще получите команда. Командите и параметрите винаги ще са верни (командите Add, Release,

HasEmptyRooms и Print винаги ще са за съществуващи клиники/любимци) с изключение на броя стаи в командата Create Clinic, който може да е всяко валидно цяло число между 1 и 101.

Изход

За всяка команда с булев връщан тип данни, получена през входа, трябва да отпечатате върнатата от нея стойност на нов ред. В случай, че метод хвърли изключение (например при опит за създаване на клиника с четен брой стаи или за добавяне на любимец, който не съществува), трябва да го прихванете и вместо това да отпечатате "Invalid Operation!". Командата Print с клиника и стая трябва да отпечатва информация за тази стая в уточнения по-горе формат. Командата Print само с клиника трябва да отпечатва информация за всяка стая в клиниката по ред на номерата.

Ограничения

- Броят команди N ще е валидно цяло число между [1...1000]; не е нужно да го проверявате изрично.
- Имената на любимци, имената на клиники и вида ще са низове, съдържащи само азбучни символи, с дължина между [1...50] символа.
- Възрастта на любимците ще е положително цяло число между [1...100].
- Клиничните стаи ще са положително цяло число между [1...101].
- Номерът на стая в команда Print винаги ще е между 1 и броя стаи в тази клиника.
- Входът ще се състои само от верни команди и те винаги ще имат правилен брой параметри.

Пример

Вход	Изход
9 Create Pet Gosho 7 Cat Create Clinic Rezovo 4 Create Clinic Rizovo 1 HasEmptyRooms Rizovo Release Rizovo Add Gosho Rizovo HasEmptyRooms Rizovo Create Pet Sharo 2 Dog Add Sharo Rizovo	Invalid Operation! True False True False False
8 Create Pet Gosho 7 Cat Create Pet Sosho 1 Cata Create Clinic Rezovo 5 Add Gosho Rezovo Add Sosho Rezovo Print Rezovo 3 Release Rezovo Print Rezovo	True True Gosho 7 Cat True Room empty Sosho 1 Cata Room empty Room empty Room empty

8. ***Обхождане на свързан списък

Трябва да напишете своя собствена опростена имплементация на стандартен Свързан Списък (Linked List), който имплементира IEnumerable<T>. Списъкът трябва да поддържа операциите Add и Remove; трябва да разкрива броя елементи, които има, със свойството Count. Методът Add трябва да добавя новия елемент в края на колекцията. Методът Remove трябва да

премахва първото появяване на елемента от началото на колекцията; ако елементът е успешно премахнат методът да връща true, ако пък елементът не е в колекцията методът трябва да върне false. Свойството Count трябва да върне броя елементи, които списъкът съдържа в момента.

Вход

В първия ред от входа ще получите число N. На всеки от следващите N редове ще получите команда в един от следните формати:

Add <number> - добавя число към вашия свързан списък.

Remove <number> - премахва първото появяване на числото от свързания списък. Ако няма такъв елемент, тази команда оставя колекцията непроменена.

Изход

Изходът трябва да съдържа точно 2 реда. На първия трябва да отпечатате резултата от извикването на свойството Count на свързания списък. На втория трябва да отпечатате всички елементи от колекцията на един ред отделени с интервали, като извикате метода foreach за колекцията.

Ограничения

- Всички числа във входа ще са валидни цели числа между [2-32...232-1].
- Всички команди получени през входа ще бъдат валидни (ще са само Add или Remove).
- Броят хора N ще е положително цяло число между [1...500].

Примери

Вход	Изход
5 Add 7 Add -50 Remove 3 Remove 7 Add 20	2 -50 20
6 Add 13 Add 4 Add 20 Add 4 Remove 4 Add 4	4 13 20 4 4

Съвет

Може да използвате упражненията за линейни структури от данни, което ще ви помогне да имплементирате своя свързан списък. Ресурсите там би трябвало да са достатъчно ръководство как да го направите.