

Упражнения: Рекурсия-1

1. Обръщане последователността на масив

Напишете програма, която обръща и отпечатва масив. Използвайте рекурсия.

Примери

Вход	Изход
1 2 3 4 5 6	6 5 4 3 2 1

2. Вложени цикли и рекурсия

Напишете програма, която симулира изпълнението на n вложени цикъла от 1 до n , която отпечатва стойностите на всичките си итерационни променливи по всяко време на един ред. Използвайте рекурсия.

Примери

Вход	Изход	Решение с вложени цикли (приемаме, че n е положително)
2	1 1 1 2 2 1 2 2	<pre>int n = 2; for (int i1 = 1; i1 <= n; i1++) { for (int i2 = 1; i2 <= n; i2++) { Console.WriteLine(\$"{i1} {i2}"); } }</pre>
3	1 1 1 1 1 2 1 1 3 1 2 1 1 2 2 ... 3 2 3 3 3 1 3 3 2 3 3 3	<pre>int n = 3; for (int i1 = 1; i1 <= n; i1++) { for (int i2 = 1; i2 <= n; i2++) { for (int i3 = 1; i3 <= n; i3++) { Console.WriteLine(\$"{i1} {i2} {i3}"); } } }</pre>

3. Комбинации с повторения

Напишете **рекурсивна** програма за генериране и отпечатване на всички **комбинации с повторения** на k елемента от набор от n елементи ($k \leq n$). В комбинациите, **редът на елементите няма значение**,

следователно (1 2) и (2 1) са една и съща комбинация, което означава, че след като получите (1 2), (2 1) вече не е валидно.

Примери

Вход	Иход	Коментари	Решение с вложени цикли
3 2	1 1 1 2 1 3 2 2 2 3 3 3	<ul style="list-style-type: none"> n=3 => имаме множество от 3 елемента {1, 2, 3} k=2 => избираме два от три елемента всеки път Повтренията са позволени, което значи, че (1 1) е валидна комбинация 	<pre>int n = 3; int k = 2; // k == 2 => 2 nested for-loops for (int i1 = 1; i1 <= n; i1++) { for (int i2 = i1; i2 <= n; i2++) { Console.WriteLine(\$"{i1} {i2}"); } }</pre>
5 3	1 1 1 1 1 2 1 1 3 1 1 4 1 1 5 1 2 2 ... 3 5 5 4 4 4 4 4 5 4 5 5 5 5 5	<p>Избираме 3 елемента от общо 5 – {1, 2, 3, 4, 5}, общо 35 комбинации</p> <p>(1 2 1) не е валидна комбинация, тъй като е същата като (1 1 2)</p>	<pre>int n = 5; int k = 3; // k == 3 => 3 вложени цикъла for (int i1 = 1; i1 <= n; i1++) { for (int i2 = i1; i2 <= n; i2++) { for (int i3 = i2; i3 <= n; i3++) { Console.WriteLine(\$"{i1} {i2} {i3}"); } } }</pre>

4. Ханойски кули

Вашата цел е да преместите всички дискове от пръчката източника на пръчката местоназначение. Има няколко **правила**:

- само един диск може да бъде преместен в даден момент
- само най-горния диск може да бъде преместен
- един диск може да бъдат поставен само върху по-голям диск, или върху празна пръчка

Step 1. Избор на подходяща структура от данни

Първо ние трябва да решим как да моделираме проблема в нашата програма. Размерът на диска може да бъде представяван от **цяло число** – колкото по-голямо е числото, толкова по-голям да е диска. Какво ще кажете за пръчките? Според правилата, описани по-горе ние може да вземем диск от върха на пръчката, или да поставим диск върху него. Това е пример за **последно влезнал-първи излезна (LIFO)**, следователно, подходяща структура да се представи един прът ще бъде стек<T>. Тъй като ние ще се съхраняваме цели числа за дискове, ние се нуждаем от **3 Stack<int>** - **източник, дестинацията и резерв**

Step 2. Настройки

Сега, ние имаме представа кои структури ще се използват, и е време за първоначална настройка. Преди решаването на пъзела за произволен брой дискове, нека го решим с 3 и използваме фиксирани конкретни стойности. С 3 дискове тя ще бъде по-лесно за следене и вземане на мерки. Първоначално местоназначението и резервните са празни. В източника, трябва да имаме числата 1, 2 и 3, 1 е на върха. Можем да използваме метода на Enumerable.Range за да получим поредица от числа като ни се предоставя начална стойност и брой елементи:

```
var range = Enumerable.Range(1, 3); // 1, 2, 3
```

Конструкторът на Stack<T> ни позволява да предадем една колекция, която ще се използва за създаване на стека. Ако ние предаваме променлив обем на конструктора, най-големия диск ще бъде на върха, което не е това, което искаме, така че можем да извикаме Reverse метод от LINQ за да се обърнат числата. Ние можем да пропуснем диапазона и да го фиксираме на 3 директно ето така:

```
Stack<int> source = new Stack<int>(Enumerable.Range(1, 3).Reverse());  
Stack<int> destination = new Stack<int>();  
Stack<int> spare = new Stack<int>();
```

Step 3. Разделяне на задачата на подзадачи

Задачата за ханойските кули се решава, като се раздели на подзадачи. Това, което ние ще се опитаме да направим е:

- 1) да преместим всички дискове от източника до местоназначението, започвайки с най-големия (долния диск)
 - a) ако диска на дъното е равен на 1, ние може просто да го преместим,
 - b) а ако дискът на дъното е по-голям от 1
 - I. трябва да преместим всички дискове по-горе го (започвайки от дъното-1) на резервния прът.
 - II. преместваме долния диск на пръчката (прътя) **местназначение(цел)**,
 - III. накрая преместете дискове сега на резервни до местоназначението (обратно в горната част на долния диск)

по същество, стъпки 1.b.i и 1.b.iii повтарят стъпка 1, единствената разлика е, че ние гледаме различни пръчки като източник, местоназначението и резервни.

Step 4. Решение

От стъпка 3 по-горе, е видно, че ще ни трябва метод, който има 4 аргументи: стойността на диска на дъното и на три пръчки.

```
private static void MoveDisks(int bottomDisk, Stack<int> source, Stack<int> destination, Stack<int> spare)
{
    // TODO
}
```

Нуждаем се от клаузата if, за да проверим дали bottomDisk == 1 (долната част на нашите рекурсия). Ако случаят е такъв, ние ще извадим елемент от източника и го поставяме на местоназначението. Можем да го направим на един ред като този:

```
if (bottomDisk == 1)
{
    destination.Push(source.Pop());
}
else
{
    // TODO
}
```

В клаузата за ИНАЧЕ, ние трябва да направим три неща: 1) преместване на всички дискове от bottomDisk - 1 от източника да резервния; 2) преместване на bottomDisk от източника до местоназначението; 3) преместване на всички дискове от bottomDisk-1 от резервния до местоназначението.

```
if (bottomDisk == 1)
{
    destination.Push(source.Pop());
}
else
{
    // TODO: Move disks on top of bottomDisk from source to spare
    destination.Push(source.Pop());
    // TODO: Move disks previously moved to spare to destination
}
```

Завършете TODOs в по-горната картина, като извикате MoveDisks рекурсивно. Ако сте направили всичко правилно, това трябва да е наред! Сега е време да го тествате.

Step 5. Проверка на решението с конкретни стойности на променливи

За да проверите това решение, нека да направим три статични стека и декларираме допълнителна променлива, която ще следи за текущия брой на предприетите стъпки.

```
private static int stepsTaken = 0;

private static Stack<int> source;
private static readonly Stack<int> destination = new Stack<int>();
private static readonly Stack<int> spare = new Stack<int>();
```

Ще ни трябва метод, който отпечатва съдържанието на всички стекове, така че ние да знаем кой диск, къде е след след всяка стъпка:

```
private static void PrintRods()
{
    Console.WriteLine("Source: {0}", string.Join(", ", source.Reverse()));
    Console.WriteLine("Destination: {0}", string.Join(", ", destination.Reverse()));
    Console.WriteLine("Spare: {0}", string.Join(", ", spare.Reverse()));
    Console.WriteLine();
}
```

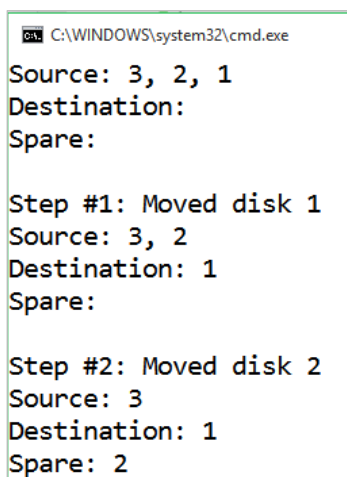
Като имаме нужните променливи и метода на PrintRods, ние можем да променят Main ето така:

```
public static void Main(string[] args)
{
    int numberOfDisks = 3;
    source = new Stack<int>(Enumerable.Range(1, numberOfDisks).Reverse());
    PrintPegs();
    MoveDisks(numberOfDisks, source, destination, spare);
}
```

В този случай ние правим статичен стек, защото от в рамките на метода на MoveDisks ние не знаем кои стека чий е. Тъй като стековете сега са статични, проверете за конфликт на имена на променливи и преименувайте параметрите на MoveDisks ако е необходимо; Тук ние само ще добавим Прът, за да разграничим статичните стекове от параметрите на метода. Сега в if и другите клаузи на MoveDisks, ние трябва да увеличим стъпките на брояч, да отпечатаме кой диск е бил преместен и да отпечатаме съдържанието на трите стека:

```
if (bottomDisk == 1)
{
    stepsTaken++;
    destinationRod.Push(sourceRod.Pop());
    Console.WriteLine($"Step #{stepsTaken}: Moved disk {bottomDisk}");
    PrintRods();
}
```

Същото се повтаря в друга клауза, разликата е рекурсивните повиквания, които правим преди и след преместването. След стартиране на програмата, сега трябва да видите всяка стъпка на процеса:



```
C:\WINDOWS\system32\cmd.exe
Source: 3, 2, 1
Destination:
Spare:

Step #1: Moved disk 1
Source: 3, 2
Destination: 1
Spare:

Step #2: Moved disk 2
Source: 3
Destination: 1
Spare: 2
```

Задачата за ханойските кули винаги отнема точно $2^n - 1$ стъпки. С $n == 3$, всички седем стъпки трябва да се покаже и в крайна сметка всички дискове трябва да свърши на местоназначението пръта. Използването на изхода на вашата програма и дебъгер, следват всяка стъпка и се опитайте да разберете как работи този рекурсивен алгоритъм. Това е много по-лесно да видите това с три дискове.

Step 6. Премахване на твърдо кодирани стойности и тестване

Ако всичко е минало добре и сте сигурни, че сте разбрали процеса, можете да замените 3 с вход от потребителя, просто прочетете няколко от конзолата. Тествайте с няколко различни стойности и се уверете, че предприетите стъпки са $2^n - 1$ и че всички дискове са успешно преместени от източника до местоназначението. Ето пълен пример с 3 диска:

Примери

Вход	Изход
3	<p>Source: 3, 2, 1</p> <p>Destination:</p> <p>Spare:</p> <p>Step #1: Moved disk 1</p> <p>Source: 3, 2</p> <p>Destination: 1</p> <p>Spare:</p> <p>Step #2: Moved disk 2</p> <p>Source: 3</p> <p>Destination: 1</p> <p>Spare: 2</p> <p>Step #3: Moved disk 1</p> <p>Source: 3</p> <p>Destination:</p> <p>Spare: 2, 1</p> <p>Step #4: Moved disk 3</p> <p>Source:</p> <p>Destination: 3</p> <p>Spare: 2, 1</p> <p>Step #5: Moved disk 1</p> <p>Source: 1</p> <p>Destination: 3</p> <p>Spare: 2</p> <p>Step #6: Moved disk 2</p> <p>Source: 1</p>

	Destination: 3, 2 Spare: Step #7: Moved disk 1 Source: Destination: 3, 2, 1 Spare:
--	---

Поздравления, това е решението на задачата за Ханойските кули с помощта на рекурсия!

5. Комбинации без повторения

Промяна на предишната програма **да пропуснете копия, например (1 1) не е валиден**.

Примери

Вход	Изход	Коментари	Решение с вложени цикли
3 2	1 2 1 3 2 3	<ul style="list-style-type: none"> n=3 => имаме множество от 3 елемента{1, 2, 3} k=2 => избираме два от три елемента Повторенията не са позволени, сиреч (1 1) не е валидна комбинация. 	<pre>int n = 3; int k = 2; // k == 2 => 2 вложени цикли for (int i1 = 1; i1 <= n; i1++) { for (int i2 = i1 + 1; i2 <= n; i2++) { Console.WriteLine(\$"{i1} {i2}"); } }</pre>
5 3	1 2 3 1 2 4 1 2 5 1 3 4 1 3 5 1 4 5 2 3 4 2 3 5 2 4 5	Избираме три от пет – {1, 2, 3, 4, 5}, всико 10 комбинции	<pre>int n = 5; int k = 3; // k == 3 => 3 вложени цикъла for (int i1 = 1; i1 <= n; i1++) { for (int i2 = i1 + 1; i2 <= n; i2++) { for (int i3 = i2 + 1; i3 <= n; i3++) {</pre>

	3 4 5		<pre> Console.WriteLine(\$"({i1} {i2} {i3})"); } } } </pre>
--	-------	--	---

6. Свързани масиви в матрица

Нека да определим свързани масиви в една матрица като област от клетки, в които **има път между всеки две клетки**.

Напишете програма, която намира **всички свързани области** в една матрица. Изведете на екрана общия брой намерени площи и на отделен ред информация за всяка от областите – позиция (горния ляв ъгъл) и размер. Подредете областите по размер (в низходящ ред), така че най-голямата площ се отпечатва първо. Ако няколко области имат същия размер, подредете ги **по своята позиция**, първа е тази, чийто горен ляв ъгъл е по-горе и/или по-вдясно на реда. Така че ако има две свързани области със същия размер, която е над и/или вляво от другата ще бъде отпечатана първо.

- На първия ред вие ще получите **броя на редовете**
- На втори ред вие ще получите **броя на колоните**
- Останалата част от вход ще бъде **действителната матрица**.

Примери

Примерно разположение	Изход																																																		
<div>4</div> <div>9</div> <table><tr><td>1</td><td>-</td><td>-</td><td>*</td><td>2</td><td>-</td><td>-</td><td>*</td><td>3</td></tr><tr><td>-</td><td>-</td><td>-</td><td>*</td><td>-</td><td>-</td><td>-</td><td>*</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>*</td><td>-</td><td>-</td><td>-</td><td>*</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>*</td><td>-</td><td>*</td><td>-</td><td>-</td></tr></table>	1	-	-	*	2	-	-	*	3	-	-	-	*	-	-	-	*	-	-	-	-	*	-	-	-	*	-	-	-	-	-	*	-	*	-	-	<div>Обща площ: 3</div> <div>Площ #1 at (0, 0), размер: 13</div> <div>Площ #2 at (0, 4), размер: 10</div> <div>Площ #3 at (0, 8), размер: 5</div>														
1	-	-	*	2	-	-	*	3																																											
-	-	-	*	-	-	-	*	-																																											
-	-	-	*	-	-	-	*	-																																											
-	-	-	-	*	-	*	-	-																																											
<div>5</div> <div>10</div> <table><tr><td>*</td><td>1</td><td></td><td>*</td><td>3</td><td></td><td></td><td>*</td><td>2</td><td></td></tr><tr><td>*</td><td></td><td></td><td>*</td><td></td><td></td><td></td><td>*</td><td></td><td></td></tr><tr><td>*</td><td></td><td></td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td></td><td></td></tr><tr><td>*</td><td></td><td></td><td>*</td><td>4</td><td></td><td></td><td>*</td><td></td><td></td></tr><tr><td>*</td><td></td><td></td><td>*</td><td></td><td></td><td></td><td>*</td><td></td><td></td></tr></table>	*	1		*	3			*	2		*			*				*			*			*	*	*	*	*			*			*	4			*			*			*				*			<div>Общо намерени области : 4</div> <div>Площ #1 at (0, 1), размер: 10</div> <div>Площ #2 at (0, 8), размер: 10</div> <div>Площ #3 at (0, 4), размер: 6</div> <div>Площ #4 at (3, 4), размер: 6</div>
*	1		*	3			*	2																																											
*			*				*																																												
*			*	*	*	*	*																																												
*			*	4			*																																												
*			*				*																																												

Подсказки

- Създаване на метод за намиране на първата „проходима“ клетка, която не е била посетена. Тя ще бъде в горния ляв ъгъл на свързаната област. Ако няма такива клетки, това означава, че всички области са били открити.
- Можете да създадете клас, който да държи информация за свързаната област (своята позиция и размер). Освен това можете да реализирате Comparable и да съхранявате всички намерени области в подредено множество.

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма **"Обучение за ИТ кариера"** на МОН за подготовка по професия "Приложен програмист".



Министерство
на образованието
и науката



Национална
програма
„Обучение за
ИТ кариера“

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).



SoftUni
Foundation

