# Упражнения: Генериране на пермутации

## Генериране на пермутации

Да се напише програма, която въвежда число и генерира всички възможни пермутации от неговите цифри.

### Пример

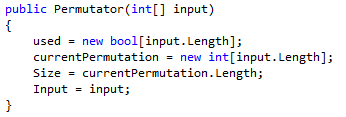
|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 123 | 123  132  213  231  312  321 |

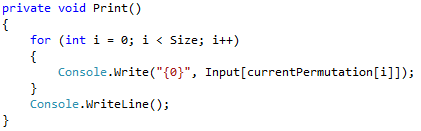
### Решение

Алгоритъмът за генериране на пермутации е удобно да се реализира чрез **backtracking** схемата. Ще използваме два помощни масива – един **bool[] used**, в който ще пазим дали числото, което се опитваме да поставим е било използвано към момента и още един **int[] currentPermutation**, в който ще пазим какво число сме поставили на позиция **i**.

За целите на задачата ще приемем, че вече сме раздробили числото към масив **int[] input**, в който всяка цифра се намира на отделен елемент.

Този масив ще подадем като параметър към **конструктора** на класа **Permutator**:



Нека да създадем метод за извеждане, който изглежда така:  


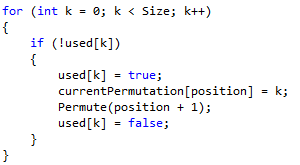
Важно е да отбележим, че самото пермутиране се случва върху масив, който съдържа числата от **0** до **Size-1** и всъщност ни дава индексите на елементите в **Input** масива. При всяко едно извикване на **Print()** елементите на **currentPermutation** имат различни стойности, които отразяват текущата пермутация.

Сега нека да реализираме според **backtracking** схемата и Permute(int **position**) метод, който ще се опитва да разположи число на индекс **position.**

Ако позицията се окаже по-голяма или равна на **Size,** то ние сме приключили с разполагането на елементи в тази пермутация и сме готови да я изведем.



В противен случай, обхождаме всичките възможни кандидати и ако текущият кандидат се окаже подходящ, т.е. не е бил използван досега, го отбелязваме като използван, разполагаме го на позиция **position** в **currentPermutation.** След това извикваме метода рекурсивно за следващата позиция. След рекурсивното връщане отбелязваме, че кандидата може да се счита за неизползван:



Важно е да уточним, че ако (част от) цифрите на входното число са еднакви, (част от) пермутациите могат да изглеждат „еднакви“. Въпреки че изглеждат еднакви, тези пермутации са различни по природа, понеже са получени от достъпването на елементи седящи на различни индекси.

Пример: 122 -> 122 122 212 221 212 221

След конструирането на класа **Permutator**, може да го ползвате така:

