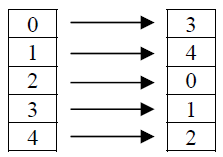
**Методы и массивы.**

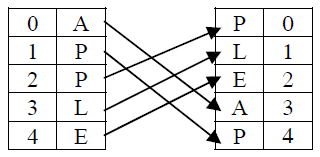
**КРИПТОГРАФИЯ НА ОСНОВЕ ПЕРЕСТАНОВОК**

Преобразование текста из открытой формы в зашифрованную и обратно является распространенной задачей шифрования. Один простой метод шифрования текста основан на математическом понятии перестановки (Permutation).

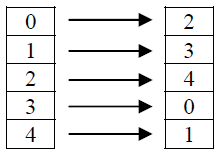
Перестановка целых чисел от 0 до N-1 — это взаимно однозначная функция, область определения и диапазон значений которой являются набором целых чисел {0, 1, 2, . . . , N-1}. Другими словами, перестановка преобразует каждое целое число в наборе {0, 1, 2, . . . , N-1} в другое целое число из того же набора, исключая возможность преобразования двух различных чисел в одно и то же число. Например, перестановка набора {0, 1, 2, 3, 4} может быть такой:



Эту перестановку можно использовать для шифрования строки из пяти символов путем перемещения каждого символа из исходной позиции в позицию, заданную перестановкой. Например, строка «**APPLE**» будет преобразована в строку «**PLEAP**». Обратите внимание, что позиции символов нумеруются с нуля:



Как расшифровать строку «**PLEAP**»? У каждой перестановки есть инверсия - другая перестановка, которая является полной противоположностью оригиналу. Для приведенной выше перестановки обратная перестановка (инверсия) будет следующей:

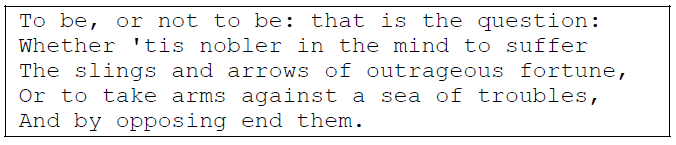


Применив обратную перестановку к зашифрованной строке «**PLEAP**», получим исходную строку «**APPLE**». Это ключевая идея шифрования на основе перестановок: **примените перестановку к исходной строке, затем к полученной строке примените инверсию перестановки и получите исходную строку.** Это означает, что мы можем использовать перестановку для шифрования текста, а затем использовать инверсию этой перестановки для расшифровки текста, чтобы получить оригинальный текст.

Требуется написать программу, которая способна как зашифровать текст (encrypting text), так и расшифровывать его (decrypting text), используя заданную перестановку символов исходного текста.

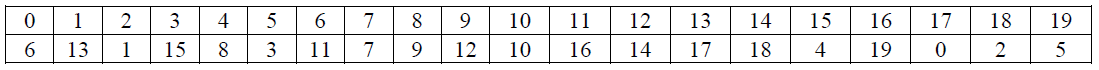
**Шифрование (Encryption):** Имея перестановку **P** и образец открытого (незашифрованного) текста **TC**, мы можем применить перестановку к открытому тексту для получения зашифрованного текста **TE**. Процесс описан ниже.

Схема шифрования, описанная выше, предполагает, что перестановка имеет ту же длину, что и исходная строка символов. Это нормально для шифрования отдельных слов, но не работает для длинных текстов, таких как:

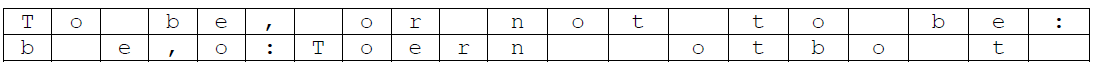


В представленном тексте 199 символов, включая символы перехода на новую строку (newline). Можно создать перестановку {0, 1, . . . , 198} и использовать ее для зашифровки этого текста. Но существует более простой и эффективный подход. Можно задать более короткую перестановку, скажем, {0, 1, . . . 19} и использовать ее для шифрования групп из 20 символов, пока мы не обработаем весь текст. Но, остается проблема последнего фрагмента текста.

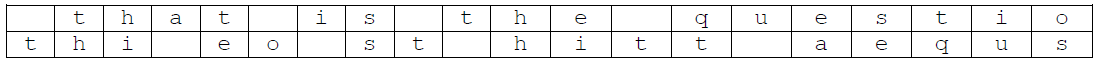
Например, берем следующую перестановку:



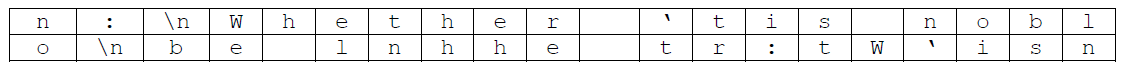
И применяем ее для первых 20 символов текста:



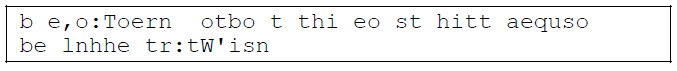
Затем следующие 20 символов и повторяем процесс:



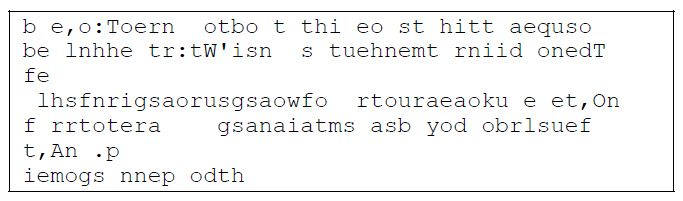
И снова для следующих 20 символов (символы перевода строки тоже переставляются):



Таким образом, рассматриваемый до сих пор текст будет зашифрован следующим образом:



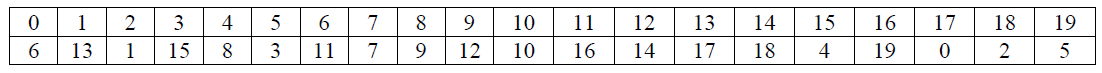
Продолжая таким образом, мы получим зашифрованный текст:



**Остается неразрешенной одна проблема!** Перестановка используется для фрагментов текста из 20 символов, но общее количество символов текста не кратно 20. Это означает, что когда мы доберемся до последнего фрагмента, то в нем будет недостаточно символов, чтобы соответствовать размеру перестановки. Поэтому для последнего фрагмента текста необходимо применить другой алгоритм.

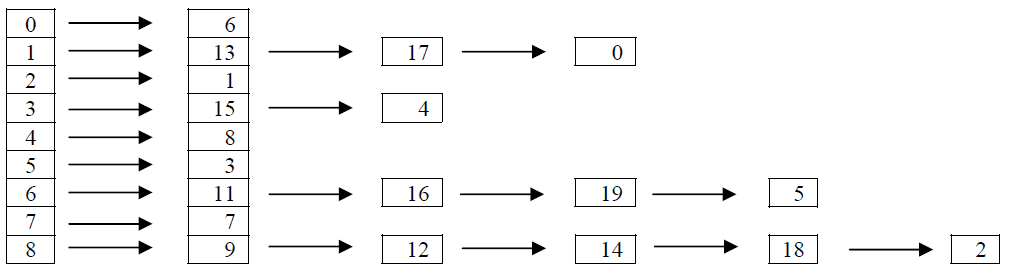
**Обработка последнего фрагмента текста**: Рассмотрим текст, представленный выше. В нем 199 символов, а размер перестановки - 20. Что означает, что после того, как мы прочитали и зашифровали 9 фрагментов текста, останется 19 символов. Это не соответствует длине перестановки. Необходимо выбрать перестановку, соответствующую длине последнего фрагмента текста.

За основу возьмем исходную перестановку {0, . . . , 20}:

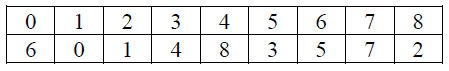


Допустим, что нам нужна перестановка длины 9 - {0, … , 8}, чтобы обработать 9 символов. Можно построить такую перестановку на основе заданной следующим образом:

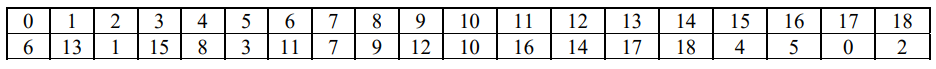
Пусть **k** принадлежит {0, . . . , 8}, а **k’** будет значением, в которое исходная перестановка отображает k. Если **k’** окажется в диапазоне {0, … , 8}, то просто сохраним это значение. Если **k’** больше 8, то смотрим, что соответствует числу **k’** в исходной перестановке. Обозначим это число **k’’**. Если **k’’** находится в диапазоне {0, . . . , 8}, затем в новой перестановке сопоставляем **k** с **k’’**. Если **k’’** больше 8, продолжаем процесс, находя значение, в которое исходная перестановка отображает **k’’**. В конце концов мы получим значение в диапазоне {0, . . . , 8}, с которым в новой перестановке сопоставим число **k.**



Таким образом, новая перестановка длины 9 выглядит следующим образом:



Теперь, когда мы создали перестановку нужного размера, мы можем применить ее для шифрования последнего фрагмента исходного текста. Последний фрагмент текста имеет длину 19 символов (включая символ перехода на новую строку), поэтому нам нужна перестановка {0, . . . , 19}. Используя подход, показанный выше, получаем перестановку:



Последний фрагмент текста будет зашифрован следующим образом:



**Расшифровка (Decryption).** Задача расшифровки текста ставится следующим образом:

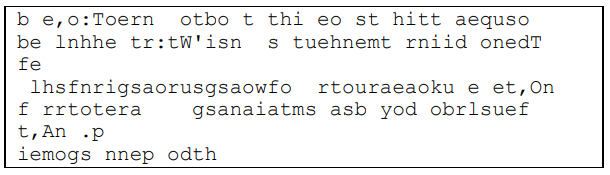
Имея зашифрованный текст **TE**, необходимо получить соответствующий открытый текст **TC**.

Процесс расшифровки основан на одном простом факте: **имея на входе открытый текст TC и перестановку P можно получить единственный зашифрованный текст TE, используя описанный выше алгоритм.**

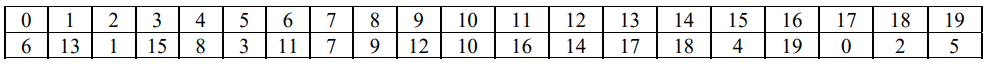
Таким образом, расшифровка текста, с математической точки зрения, является обратной операции шифрования.

Проблему расшифровки можно решить очень простым способом. Каждая перестановка **P** имеет обратную перестановку, обозначаемую **P-1**. Если мы знаем **P-1**, мы можем расшифровать зашифрованный текст **TE**, применив обратную перестановку к **TE** точно так же, как исходная перестановка **P** была применена для создания зашифрованного текста **TE**. Зная **P**, можно легко найти **P-1**. Если **P** отображает i в j, то **P-1** отображает j в i.

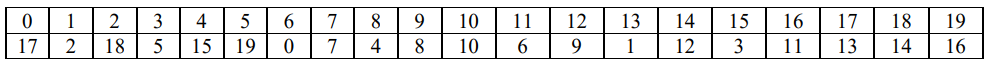
Давайте еще раз взглянем на зашифрованную версию текста:



Этот зашифрованный текст получен с использованием перестановки **P**:



Обратная перестановка **P-1** будет следующей:



Применив обратную перестановку к первому фрагменту зашифрованного текста, получим:



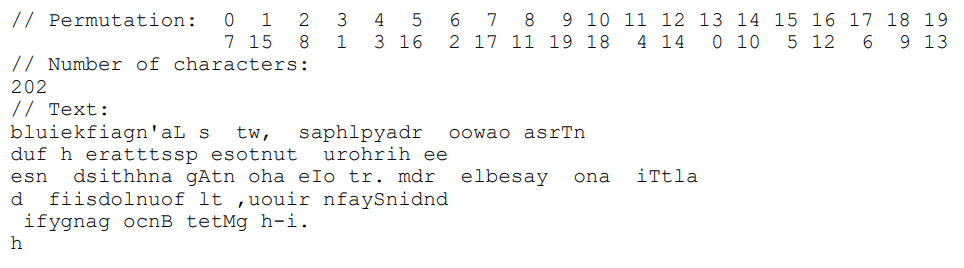
Продолжая применять обратную перестановку к следующим фрагментам зашифрованного текста, получим исходный текст.

**Формат входного файла и пример:**

Ваша программа должна считывать данные из входного файла Scrambled.txt. Первые две строки входного файла задают перестановку, которая использовалась для шифрования текста. Первую строку рассматривайте как комментарий (игнорируйте ее) и считывайте значения во второй строке в массив. Перестановка будет всегда иметь длину 20.

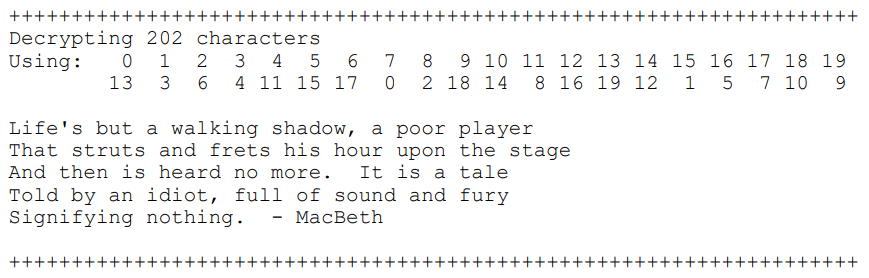
Следующие две строки задают количество символов в тексте (включая пробелы и символы перехода на новую строку). Первую строку игнорируем (это комментарий), вторую - считываем.

Далее, после строки комментария (//Text), следует текст. После последнего символа текста - символ перехода на новую строку (не считается частью текста).



**Формат выходного файла и пример:**

Ваша программа должна записать выходные данные в файл Unscrambled.txt

****

Первая строка - строка символов разделителей (++++++++++++++++++++++++++++++++).

Вторая строка - информация о размере зашифрованного текста: Decrypting N characters.

Строки 3 и 4 - перестановка, которая использовалась для расшифровки текста, обратная исходной.

Строка 5 - пустая.

Следующие строки - расшифрованный текст.

Протестируйте свое решение на предложенных файлах входных и выходных данных: 1.Scrumbled.txt, 1.Unscrumbled.txt, 2.Scrumbled.txt, 2.Unscrumbled.txt, 3.Scrumbled.txt, 3.Unscrumbled.txt.

**Замечания по реализации:** Используйте массивы. Реализуйте методы для решения отдельных подзадач. Метод Main() должен содержать не более 30 строк кода (исполняемых операторов и вызовов методов). Пустые строки, строки объявления констант и переменных, комментарии - не учитываются. Предполагается реализация, как минимум 6 методов.

Параметры методов должны передаваться надлежащим образом. Используйте передачу по ссылке только тогда, когда вызываемой функции необходимо изменить передаваемый параметр. Используйте именованные константы вместо переменных, где это уместно.