**Лабораторная работа № 32**

**Тема Алгоритмы генерации комбинаторных объектов**

**Теоретические сведения**

***Перестановка* –** это комбинация элементов из N разных элементов взятых в определенном порядке. В перестановке важен порядок следования элементов, и в перестановке должны быть задействованы все N элементов.  
  
*Задача*: Найти все возможные перестановки для последовательности чисел 1, 2, 3.

Решение: существуют следующие перестановки:  
  
1: 1 2 3  
2: 1 3 2  
3: 2 1 3  
4: 2 3 1  
5: 3 1 2  
6: 3 2 1

1. **Перестановки без повторений**

Количество перестановок для N различных элементов составляет N!. Действительно:

* на первое место может быть помещен любой из N элементов (всего вариантов N),
* на вторую позицию может быть помещен любой из оставшихся (N-1) элементов (итого вариантов N·(N-1)),
* если продолжить данную последовательность для всех N мест, то получим: N·(N-1)·(N-2)· ... ·1, то есть всего N! перестановок.

*Задача:*

Получить всех перестановок чисел 1...N (то есть последовательности длины N), где каждое из чисел входит ровно по 1 разу. Существует множество вариантов порядка получения перестановок. Однако наиболее часто решается задача генерации перестановок в *лексикографическом*порядке (см. пример выше). При этом все перестановки сортируются сначала по первому числу, затем по второму и т.д. в порядке возрастания. Таким образом, первой будет перестановка 1 2 ... N, а последней - N N-1 ... 1.  
  
*Алгоритм решения задачи.*

Дана исходная последовательность чисел. Для получения каждой следующей перестановки необходимо выполнить следующие шаги:

* Необходимо просмотреть текущую перестановку справа налево и при этом следить за тем, чтобы каждый следующий элемент перестановки (элемент с большим номером) был не более чем предыдущий (элемент с меньшим номером). Как только данное соотношение будет нарушено необходимо остановиться и отметить текущее число (позиция 1).
* Снова просмотреть пройденный путь справа налево пока не дойдем до первого числа, которое больше чем отмеченное на предыдущем шаге.
* Поменять местами два полученных элемента.
* Теперь в части массива, которая размещена справа от позиции 1 надо отсортировать все числа в порядке возрастания. Поскольку до этого они все были уже записаны в порядке убывания необходимо эту часть подпоследовательность просто перевернуть.

Таким образом мы получим новую последовательность, которая будет рассматриваться в качестве исходной на следующем шаге.

***Реализация на С++***

#include <iostream>  
using namespace std;  
void swap(int \*a, int i, int j)  
{  
  int s = a[i];  
  a[i] = a[j];  
  a[j] = s;  
}  
bool NextSet(int \*a, int n)  
{  
  int j = n - 2;  
  while (j != -1 && a[j] >= a[j + 1]) j--;  
  if (j == -1)  
    return false; // больше перестановок нет  
  int k = n - 1;  
  while (a[j] >= a[k]) k--;  
  swap(a, j, k);  
  int l = j + 1, r = n - 1; // сортируем оставшуюся часть последовательности  
  while (l<r)  
    swap(a, l++, r--);  
  return true;  
}  
void Print(int \*a, int n)  // вывод перестановки  
{  
  static int num = 1; // номер перестановки  
  cout.width(3); // ширина поля вывода номера перестановки  
  cout << num++ << ": ";  
  for (int i = 0; i < n; i++)  
    cout << a[i] << " ";

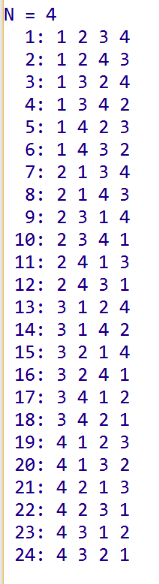
cout << endl;  
}  
int main()   
{

int n, \*a;  
  cout << "N = ";  
  cin >> n;  
  a = new int[n];  
  for (int i = 0; i < n; i++)  
    a[i] = i + 1;  
  Print(a, n);  
 while (NextSet(a, n))  
    Print(a, n);

cin.get();  
  return 0;

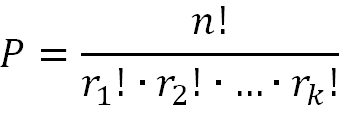
}

Результат:



1. **Перестановки с повторениями**

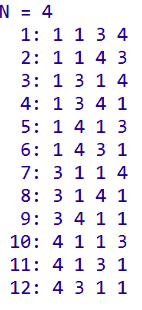
Особого внимания заслуживает задача генерации перестановок **N** элементов в случае если элементы последовательности могут повторяться. Допустим, исходная последовательность состоит из элементов **n1, n2... nk**, где элемент **n1** повторяется **r1** раз, **n2** повторяется **r2** раз и т.д. При этом **n1+n2+...+nk=N**.  Если мы будем считать все **n1+n2+...+nk** элементов перестановки с повторениями различными, то всего различных вариантов перестановок (**n1+n2+...+nk)!** . Однако среди этих перестановок не все различны. В самом деле, все **r1** элементов **n1** мы можем переставлять местами друг с другом, и от этого перестановка не изменится. Точно так же, можем переставлять элементы **n2**, **n3** и т. д. В итоге имеем **r1!** вариантов записи одной и той же перестановки с различным расположением повторяющихся элементов **n1**. Таким образом, всякая перестановка может быть записана **r1!·r2!·...·rk!** способами. Следовательно, число различных перестановок с повторениями равно:



Для генерации перестановок с повторениями можно использовать алгоритм генерации перестановок без повторений, приведенный выше. Введем повторяющийся элемент в массив a. Ниже приведен код программы для генерации перестановок с повторениями (изменен только код функции main()).

#include <iostream>  
using namespace std;  
void swap(int \*a, int i, int j)  
{  
  int s = a[i];  
  a[i] = a[j];  
  a[j] = s;  
}  
bool NextSet(int \*a, int n)  
{  
  int j = n - 2;  
  while (j != -1 && a[j] >= a[j + 1]) j--;  
  if (j == -1)  
    return false; // больше перестановок нет  
  int k = n - 1;  
  while (a[j] >= a[k]) k--;  
  swap(a, j, k);  
  int l = j + 1, r = n - 1; // сортируем оставшуюся часть последовательности  
  while (l<r)  
    swap(a, l++, r--);  
  return true;  
}  
void Print(int \*a, int n)  // вывод перестановки  
{  
  static int num = 1; // номер перестановки  
  cout.width(3); // ширина поля вывода номера перестановки  
  cout << num++ << ": ";  
  for (int i = 0; i < n; i++)  
    cout << a[i] << " ";  
  cout << endl;  
}  
int main()   
{  
  int n, \*a;  
  cout << "N = ";  
  cin >> n;  
  a = new int[n];  
  for (int i = 0; i < n; i++)  
    a[i] = i + 1;  
  a[1] = 1; // повторяющийся элемент  
  Print(a, n);  
  while (NextSet(a, n))  
    Print(a, n);  
  cin.get(); cin.get();  
  return 0;  
}

Результат работы приведенного выше алгоритма:



И наконец, в качестве сюрприза пример на Прологе (привет от Игоря Ивановича ☺)

## Генератор перестановок на Прологе

Для полноты изложения приведу здесь генератор перестановок на Прологе, который можно найти практически в любой книге по Прологу. Этот генератор состоит из двух предикатов: предикат position(L1,X,L2) для вставки элемента X в все позиции списка L1 и предикат transposition, который собственно генерирует все перестановки.

domains  
  i=integer  
  l=i\*  
predicates  
  position(l,i,l)  
  transposition(l,l)  
clauses  
  position(L,X,[X|L]).  
  position([H|T1],X,[H|T2]):-position(T1,X,T2).  
  
  transposition([],[]).  
  transposition([H|T],L):-  
    transposition(T,T1),  
    position(T1,H,L).

Посмотрим на результат генерации:

?-transposition([1,2,3],X)  
X=[1,2,3]  
X=[2,1,3]  
X=[2,3,1]  
X=[1,3,2]  
X=[3,1,2]  
X=[3,2,1]  
6 Solutions

Как видим, порядок не антилексикографический, а случайный

Задание 1. Реализовать алгоритма генерации перестановок Джонсона-Троттера :

**Описание алгоритма**

* На русском яз. + псевдокод: [https://books.google.ru/books?id=HIaf7DSPtl0C&pg=PA228&lpg=PA228&dq=алгоритм+джонсона+троттера&source=bl&ots=YOhS\_RzlkF&sig=754ZHiufVsdeiLr8fxQpuSsHjaU&hl=ru&sa=X&sqi=2&ved=0ahUKEwjxo83l5ffTAhVsJ5oKHTasBSYQ6AEINDAC#v=onepage&q=алгоритм%20джонсона%20троттера&f=false](https://books.google.ru/books?id=HIaf7DSPtl0C&pg=PA228&lpg=PA228&dq=алгоритм+джонсона+троттера&source=bl&ots=YOhS_RzlkF&sig=754ZHiufVsdeiLr8fxQpuSsHjaU&hl=ru&sa=X&sqi=2&ved=0ahUKEwjxo83l5ffTAhVsJ5oKHTasBSYQ6AEINDAC" \l "v=onepage&q=алгоритм%20джонсона%20троттера&f=false)
* На англ.яз с картинками- понятно: https://mathlesstraveled.com/2013/01/03/the-steinhaus-johnson-trotter-algorithm/

**Учебное пособие можно взять здесь** http://fit.nsu.ru/data\_/courses/niu/daio\_komb\_alg\_uchpos.pdf