Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пензенский Государственный Университет»

**Факультет Вычислительной техники  
Кафедра «Вычислительная техника»**

**КУРСОВАЯ РАБОТА**

По дисциплине «Логика и основы алгоритмизации в инженерных задачах»

На тему: «Реализация алгоритма генерирования всех перестановок заданного множества в лексикографическом порядке»

Выполнил:

студент 2-го курса

группы 20ВВ4

Айкашев М.А.

Научный руководитель:

Юрова О.В.

Пензаы 2021

**Оглавление**

[**Введение** 3](#_Toc91443310)

[**Постановка задачи** 4](#_Toc91443311)

[**Теоретическая часть программы** 5](#_Toc91443312)

[**Описание алгоритма программы** 7](#_Toc91443313)

[**Ручной просчёт алгоритма** 10](#_Toc91443314)

[**Тестирование программы** 11](#_Toc91443315)

[**Заключение** 13](#_Toc91443316)

[**Список литературы** 14](#_Toc91443317)

[**Листинг программы** 15](#_Toc91443318)

# **Введение**

На рубеже третьего тысячелетия мировой общественностью четко осознаны две ключевые характеристики эффективной системы современного высшего образования. Это – его фундаментализация и гуманитаризация. Они одинаково важны в подготовке специалистов любого профиля – гуманитарного, естественнонаучного, инженерно-технического. Их реализация широко признана важнейшим условием подготовки специалистов, квалификация которых соответствовала бы объективным требованиям современной эпохи с её беспрецедентным динамизмом, частыми сменами господствующих научных концепций, недопустимостью узко-технократического подхода к решению сложных проблем, сложным взаимодействием разных типов национальных культур и др.

Данная проблема имеет трудности в решении. Она дробится на ряд более частных, соответствующих особенностям высшего образования того или иного профиля с той или иной специализацией. Поэтому решения логических задач крайне важны в наше непростое время.

# **Постановка задачи**

Задача данной курсовой работы состоит в следующем: написать программу генерации перестановок заданного множества в лексикографическом порядке.

Необходимо найти наиболее подходящий язык программирования для написания программы генерации перестановок, а также написать оптимальный код для реализации алгоритма. Не менее важной задачей является создание удобного, быстрого интерфейса программы, чтобы пользователь не испытывал проблем при работе.

**Необходимо определить, какой тип данных использовать.** Он наиболее подходящий для реализации массива элементов заданного множества. Это необходимо для структуризации данных, а также при обработке данных в коде программы.

**Многофункциональность программы.** Необходимо поделить программу на несколько функций, чтобы увеличить быстродействие программы.

**Программа должна работать в графическом или текстовом режиме видеоподсистемы.** Необходимо определиться с типом интерфейса и с элементами управления. В данном случае будет использована консоль Windows.

**Устройства ввода – вывода – клавиатура и мышь.** Необходимо научится анализировать и различать события, возникающие от работы с клавиатурой и мышью. Необходимо однозначно идентифицировать и выполнять те или иные действия в зависимости от действия пользователя. Это необходимо для легкого использования программы.

**Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.** Необходимо изучить технологию построения меню. Это необходимо для создания интуитивно понятного интерфейса.

# **Теоретическая часть программы**

Перестановкой n-элементного множества X называется упорядоченный набор длины n, составленный из попарно различных элементов множества X. Опишем некоторые методы генерирования последовательности всех n! перестановок n-элементного множества. Не нарушая общности будем рассматривать не исходное множество Х, а множество А= {1, 2, …, n} – множество индексов элементов, т.к. между множеством элементов из Х и множеством индексов из А существует взаимно однозначное соответствие, которое задается в виде: ↔ i.

Метод лексикографической перестановки легче всего понять, если в качестве переставляемых элементов взять числа 1, …, n. На множестве всех перестановок определим лексикографический порядок:

<x1, x2, …, xn> < <y1, y2, …, yn> ⇔ существует k≥1, такое что xk <yk и xp= yp для каждого p <k.

Если вместо чисел 1,2, …, n взять буквы а,б, …,р с естественным порядком а<б<в<…<р, то лексикографический порядок определяет стандартную последовательность, в которой слова длины n появляются в словаре.

Для написания программы, реализующей данный алгоритм, используется среда Visual Studio - линейка продуктов компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), включающих [интегрированную среду разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения и ряд других инструментальных средств. Данная линейка позволяет разрабатывать как [консольные](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [приложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), так и приложения с [графическим интерфейсом](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), в том числе с поддержкой технологии [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms). Была выбрана версия программы от 2019 года, поскольку она отличается стабильностью и высокой скоростью работы с создаваемым приложением.

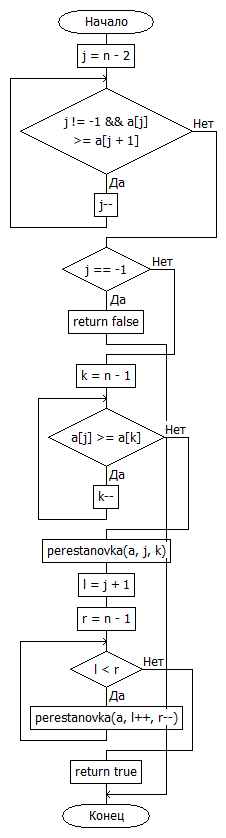
В качестве языка программирования был выбран Cи. Рост популярности Си, накапливающиеся с годами изменения, создание компиляторов коллективами разработчиков, ранее не причастных к проектированию языка, — все это послужило стимулом к более точному и отвечающему времени определению языка. Он обладает высокой скоростью работы и надёжным компилятором, что позволяет программисту создавать стабильные приложения, используя данный язык. Благодаря широким возможностям и универсальности для решения многих задач он удобнее и эффективнее, чем предположительно более мощные языки.

# **Описание алгоритма программы**

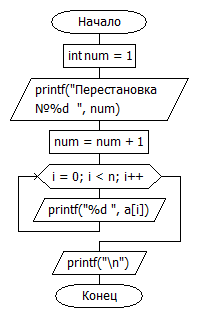
Программа работает следующим образом: пользователь вводит количество элементов заданного множества. Затем он вводит непосредственно сами элементы. Далее программа вызывает процедуру перестановки в лексикографическом порядке. Проверяются следующие условия: количество перестановок и сравнение элементов. Если у нас есть возможные перестановки и остались не переставленные элементы, то программа переставляет элементы методом пузырька (отдельная процедура). Пользователю предоставляется увидеть перестановки на экране, так как после каждой перестановки множество выводится на экран монитора.

Особенностью программы является вспомогательный массив для заданного множества. После ввода элементов программа сохраняет их в отдельный вспомогательный массив. Это позволяет делать любые перестановки, используя исходный набор данных и не покидая основную программу.

Ниже на скриншотах представлены блок-схемы основных процедур программы: процесс перестановки и вывод массива на экран. Также представлена таблица работы меню программы.

****

**Рис.1. Алгоритм генерации перестановок заданного множества в лексикографическом порядке.**

****

**Рис.2. Алгоритм вывода заданного множества на экран.**

****

**Рис.3. Алгоритм перестановки элементов.**

**Таблица 1 – Описание действий программы**

|  |  |  |
| --- | --- | --- |
| Клавиша | Действие пользователя при её нажатии | Описание действия |
| 1 | Выбран лексикографический способ перестановки | На экран выводятся всевозможные перестановки элементов. |
| 2 | Выбран пункт задания нового множества | Пользователь заново задаёт элементы множества |
| 3 | Выбран пункт изменения элементов множества при неизменном порядке элементов | Пользователь задаёт элементы множества, не меняя порядок множества |
| 4 | Выбран пункт выхода из программы | Пользователь покидает программу |

# **Ручной просчёт алгоритма**

Ручной просчёт алгоритма можно сделать по следующему алгоритму:

1. Выделяем хвост текущей перестановки;
2. Если он не совпадает со всей перестановкой, то ищем в хвосте первый с конца перестановки элемент, больший элемента перестановки, расположенного непосредственно перед ее хвостом (если перестановка совпадает со своим хвостом, то она является заключительной во всей генерации);
3. Меняем местами элемент, найденный в предыдущем пункте, с элементом, расположенным непосредственно перед хвостом перестановки;
4. Располагаем все элементы, преобразованного в пункте 3 хвоста перестановки, в обратном порядке (инвертирование преобразованного хвоста перестановки).

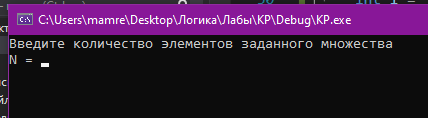
В качестве примера приведём множество третьего порядка 1 2 3. По формуле n! количество перестановок у нас будет шесть. Далее, следуя алгоритму выше, производим перестановку:

1. 1 2 3 – хвост множества равен 3, сравниваем со 2-ыми элементом и выполняем перестановку.
2. 1 3 2– хвост множества равен 2, сравниваем со 1-ыми элементом, так как 3>2, и выполняем перестановку.
3. 2 1 3– хвост множества равен 3, сравниваем со 2-ыми элементом и выполняем перестановку.
4. 2 3 1 - хвост множества равен 1, перестанавливаем 1-ый элемент в конец множества.
5. 3 1 2 – хвост множества равен 2, сравниваем со 2-ыми элементом и выполняем перестановку.
6. 3 2 1 – результат перестановки.

# **Тестирование программы**

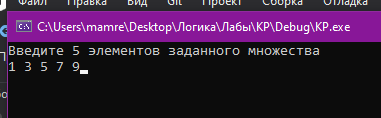
Для того чтобы проверить программу на правильность выполнения перестановок, проведём её тестирование на примере 5-ти элементов.

Сначала введём порядок заданного множества.



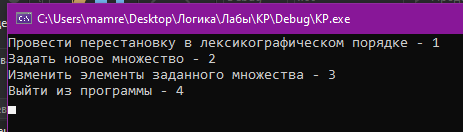
**Рис.4. Ввод порядка элементов заданного множества.**

После ввода порядка программа запросит непосредственно элементы множества.



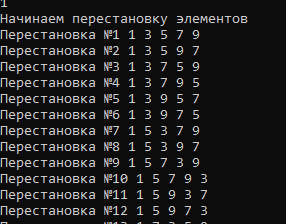
**Рис.5. Ввод элементов заданного множества.**

Далее необходимо выбрать порядок, в котором элементы будут переставлены. Пользователю также предоставляется возможность изменить элементы действующего множества или же задать новый порядок множества.

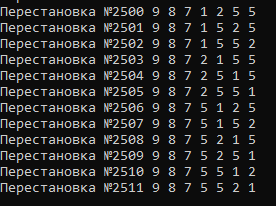


**Рис.6. Меню пользователя.**

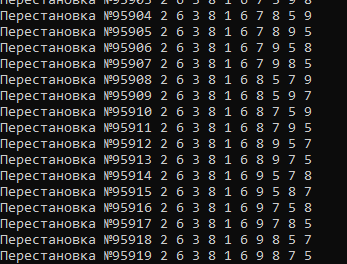
После выбора программа начнёт перестановки элементов множества в порядке, который выбрал пользователь.



**Рис.7. Перестановка в лексикографическом порядке.**



**Рис.10. Пример перестановки множества элементов при порядке n=7.**



**Рис.11. Фрагмент перестановки множества элементов при порядке n=10.**

# **Заключение**

В ходе выполнения курсовой работы была написана программа, реализующая алгоритм перестановки заданного множества лексикографическим способом.

Были закреплены навыки написания программ на языке программирования Си, также получены теоретические и практические знания в области логики и генерации перестановок множества.

Несмотря на эффективную реализацию алгоритма в рамках курсовой работы планируется доработать основную программу. Планируется в следующей версии добавить возможность редактировать каждый элемент множества, чтобы была возможность наиболее точно проверить алгоритм в исключительных случаях. Помимо этого, планируется реализовать возможность записи результата работы генерации в отдельный файл, а также считывание оттуда.

# **Список литературы**

1. Абачиев, С.К. Формальная логика с элементами теории познания: учебник для вузов / С. К. Абачиев. - Ростов-на-Дону: Феникс, 2018.- 635 с.
2. Берков, В. Ф. Логика. Учебное пособие / В.Ф. Берков. - М.: ТетраСистемс, 2018. - 208 c.
3. Брайан Керниган, Деннис Ритчи , Язык программирования Си – 3-е издание, Москва: издательский дом Уильямс, 2017. – 253 с.
4. Григорьев, А.А. Методы и алгоритмы обработки данных: Учебное пособие / А.А. Григорьев. - М.: Инфра-М, 2018. - 384 c.
5. Колдаев, В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев. - М.: Форум, 2015. - 352 c.
6. Мартынов, Б.М. Программирование для Windows на С/С++. В 2-х томах. Том 1 / Б.М. Мартынов. - М.: Бином-Пресс, 2013. - 528 c.
7. Одерски, М. Scala. Профессиональное программирование / М. Одерски, Л. Спун, Б. Веннерс. - СПб.: Питер, 2018. - 448 c.
8. Раго, С. UNIX. Профессиональное программирование / С. Раго, У. Стивенс. - СПб.: Символ-плюс, 2015. - 1104 c.
9. Семакин, И.Г. Основы алгоритмизации и программирования. Практикум: Учебное пособие / И.Г. Семакин. - М.: Academia, 2017. - 328 c.
10. Серкова, Е.Г. Основы алгоритмизации и программирования: практикум / Е.Г. Серкова. - РнД: Феникс, 2019. - 189 c.

# **Листинг программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <locale>

#include <string.h>

#include <Windows.h>

using namespace std;

void perestanovka(int\* a, int i, int j)

{

int s = a[i];

a[i] = a[j];

a[j] = s;

}

bool leks(int\* a, int n)

{

int j = n - 2;

while (j != -1 && a[j] >= a[j + 1]) j--;

if (j == -1)

return false;

int k = n - 1;

while (a[j] >= a[k]) k--;

perestanovka(a, j, k);

int l = j + 1;

int r = n - 1;

while (l < r)

perestanovka(a, l++, r--);

return true;

}

void print(int\* a, int n)

{

static int num = 1;

printf("Перестановка №%d ", num);

num = num + 1;

for (int i = 0; i < n; i++) {

printf("%d ", a[i]);

}

printf("\n");

}

void sohr(int\* m, int\* a, int n) {

for (int i = 0; i < n; i++) {

a[i] = m[i];

}

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n, \* a;

char\* b;

int\* m;

char\* m1;

char choice;

A: system("cls");

printf("Введите количество элементов заданного множества\nN = ");

scanf("%d", &n);

a = new int[n];

m = new int[n];

m1 = new char[n];

b = new char[n];

B: system("cls");

printf("Введите %d элементов заданного множества\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &a[i]);

m[i] = a[i];

}

system("cls");

C: sohr(m, a, n);

system("cls");

printf("Провести перестановку в лексикографическом порядке - 1\n");

printf("Задать новое множество - 2\n");

printf("Изменить элементы заданного множества - 3\n");

printf("Выйти из программы - 4\n");

scanf("%s", &choice);

switch (choice) {

case '1':

system("cls");

printf("Начинаем перестановку элементов\n");

print(a, n);

while (leks(a, n))

print(a, n); //выводим элементы

\_getch();

goto C;

break;

case '2':

system("cls");

goto A;

break;

case '3':

system("cls");

goto B;

break;

case '4':

system("cls");

exit(0);

break;

default:

system("cls");

goto C;

break;

}

getchar();

return 0;

}