Министерство образования Российской Федерации

Пензенский государственный университет

**Кафедра «Вычислительная техника»**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «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

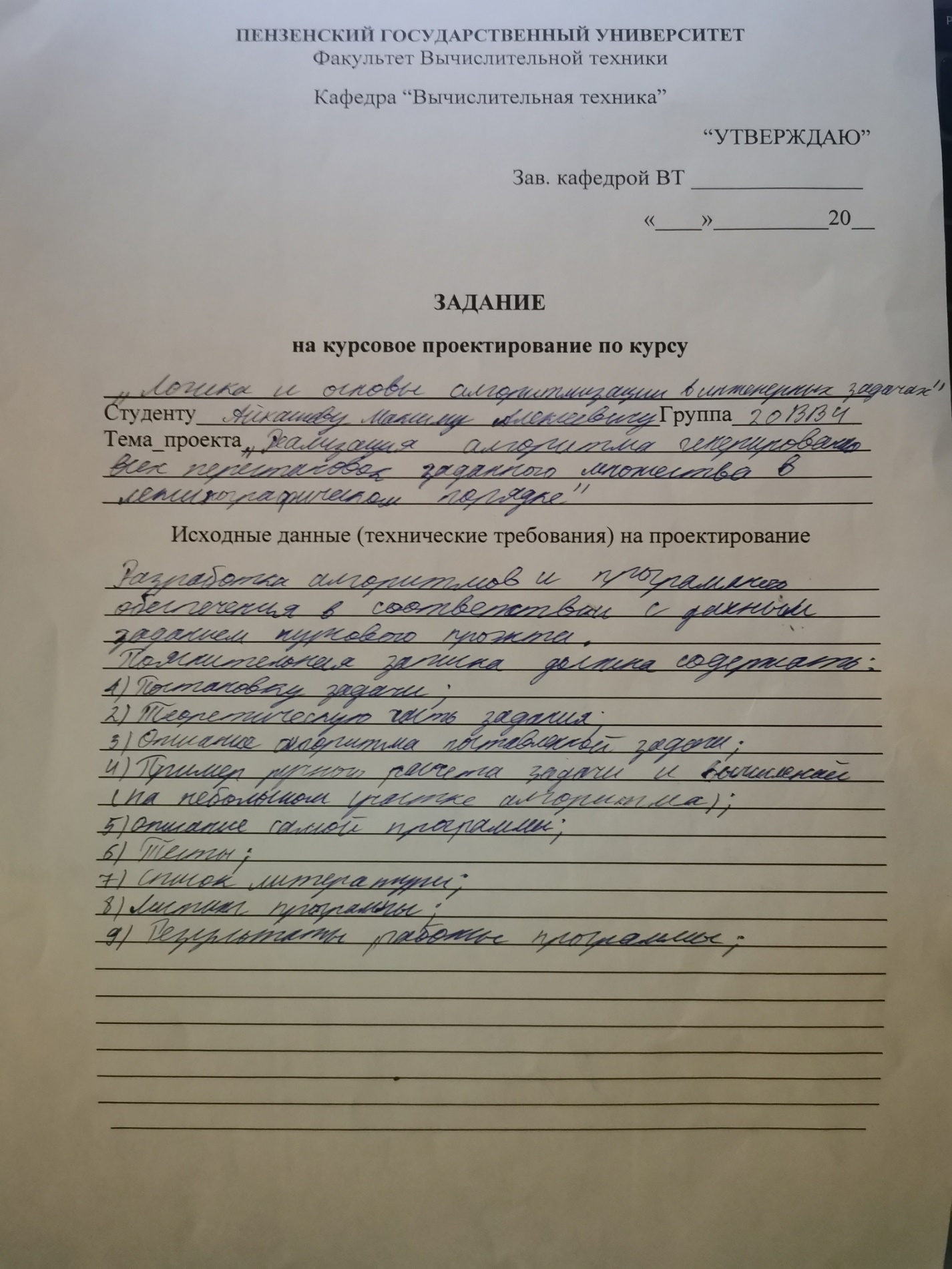
Айкашев М.А.

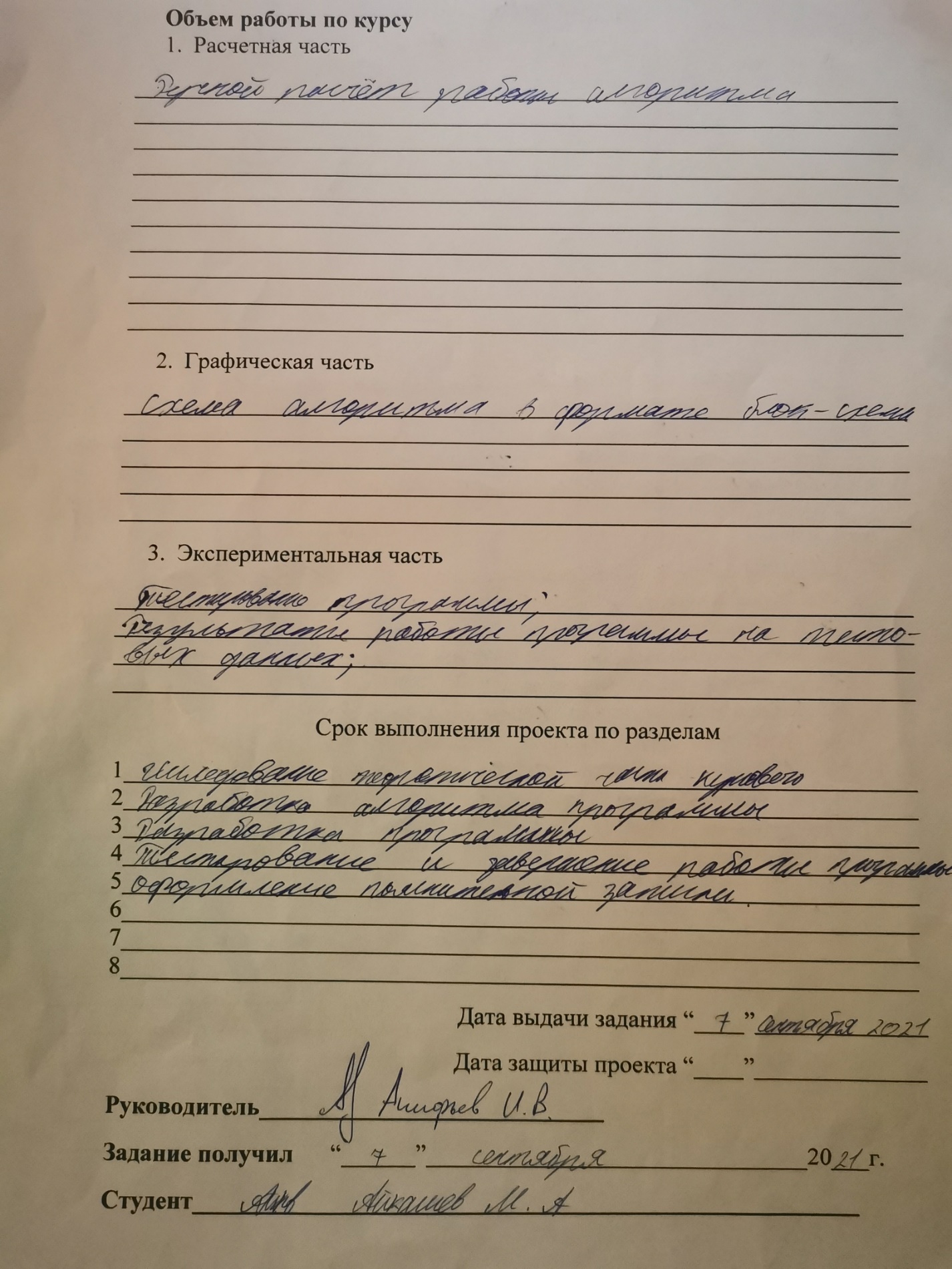
Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2021





**Содержание**

[Реферат 5](#_Toc91518202)

[Введение 6](#_Toc91518203)

1. [Постановка задачи 7](#_Toc91518204)
2. [Теоретическая часть программы 8](#_Toc91518205)
3. [Описание алгоритма программы 10](#_Toc91518206)
4. [Ручной просчёт алгоритма 13](#_Toc91518207)
5. [Тестирование программы 14](#_Toc91518208)

[Заключение 16](#_Toc91518209)

[Список литературы 17](#_Toc91518210)

[Приложение А. Листинг программы 18](#_Toc91518211)

# **Реферат**

Отчет 20 стр, 9 рисунков.

МНОЖЕСТВО ЭЛЕМЕНТОВ, ПЕРЕСТАНОВКА, ЛЕКСИКОГРАФИЧЕСКИЙ ПОРЯДОК.

Цель исследования – разработка программы, способная осуществить перестановку в лексикографическом порядке, используя алгоритм Нарайаны.

В работе рассмотрен способ получения всех перестановок в лексикографическом порядке. Установлено, что с помощью данного алгоритма можно получить список всех перестановок.

# **Введение**

Лексикографический порядок представляет собой обобщение алфавитного порядка из словарей в последовательность упорядоченных символов или, в более общем случае, из элементов полностью упорядоченного множества.

Существует несколько вариантов и обобщений лексикографической упорядоченности. Один вариант применяется к последовательностям разной длины путем сравнения длин последовательностей перед рассмотрением их элементов.

Другой вариант, широко используемый в комбинаторике, упорядочивает подмножества данного конечного множества путем присвоения общего порядка конечному набору и преобразования подмножеств в возрастающие последовательности, к которым применяется лексикографический порядок.

Обобщение определяет порядок на декартово произведение из частично упорядоченных множеств; этот порядок является полным, если и только если все факторы декартова произведения полностью упорядочены.

# **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 [3].

**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 [1]. 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 (Рисунок 1). 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 [1].

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 [2].

Ниже на рисунках представлены блок-схемы основных процедур программы: процесс перестановки и вывод массива на экран (Рисунок 2-3). Также представлена таблица работы меню программы (Таблица 1).

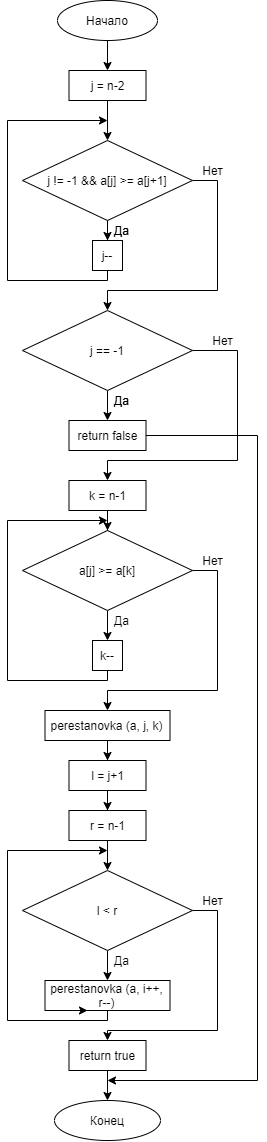
****

Рисунок 1 - Алгоритм генерации перестановок заданного множества в лексикографическом порядке

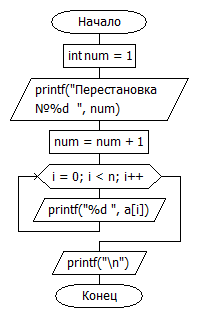
****

Рисунок 2 - Алгоритм вывода заданного множества на экран

****

Рисунок 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 (Рисунок 4-7).

Сначала введём порядок заданного множества.

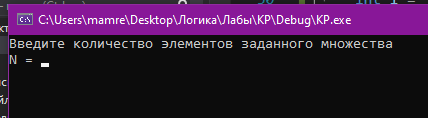


Рисунок 4 - Ввод порядка элементов заданного множества

После ввода порядка программа запросит непосредственно элементы множества.

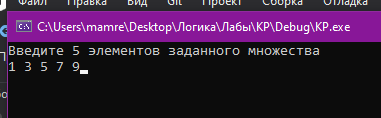


Рисунок 5 - Ввод элементов заданного множества

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

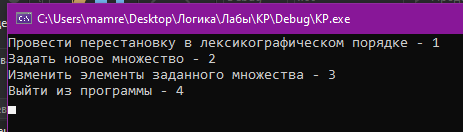


Рисунок 6 - Меню пользователя

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

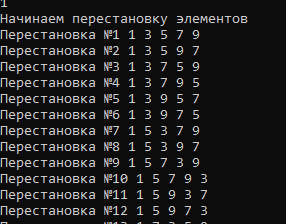


Рисунок 7 - Перестановка в лексикографическом порядке

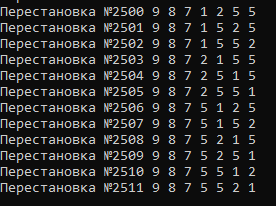


Рисунок 8 - Пример перестановки множества элементов при порядке n=7

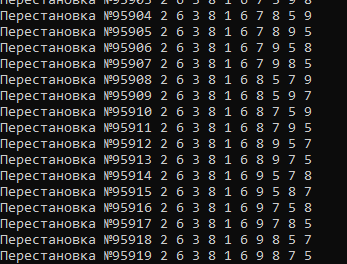


Рисунок 9 - Фрагмент перестановки множества элементов при порядке n=10

# **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.

# **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.

# **Приложение А. 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) //Алгоритм Нарайаны

{

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;

char vib;

A: system("cls");

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

scanf("%d", &n);

a = new int[n];

m = new int[n];

m1 = new char[n];

b = new char[n];

B: system("cls");

printf("Выберите способ ввода:\n");

printf("Ввести вручную - 1\n");

printf("Ввести случайно - 2\n");

scanf("%s", &vib);

switch (vib) {

case '1':

system("cls");

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

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

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

m[i] = a[i];

}

system("cls");

break;

case '2':

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

a[i] = rand() % 10;

m[i] = a[i];

}

system("cls");

printf("Полученный массив: ");

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

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

}

printf("\n");

\_getch();

break;

default: //Возврат

system("cls");

goto B;

break;

}

C: sohr(m, a, n);

system("cls");

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

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': //Перестановка

system("cls");

printf("Начинаем перестановку элементов\n");

print(a, n);

while (leks(a, n))

print(a, n); //Вывод элементов

FILE\* fout;

fout = fopen("D:\\out.txt", "w"); //Запись в файл

if (fout) {

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

fprintf(fout, "%d ", a[i]);

}

}

\_getch();

goto C;

break;

case '2': //Новое множество

system("cls");

goto A;

break;

case '3': //Перезаполнение множества

system("cls");

goto B;

break;

case '4': //Выход

system("cls");

exit(0);

break;

default: //Возврат

system("cls");

goto C;

break;

}

getchar();

return 0;

}