|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №1  по дисциплине «Языки программирования»  Вариант 18 | | |
|  | | Работу выполнил  студент группы ИТ-17,18-2024 1 курса  Кужлева Л.В.\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «14» \_\_января\_\_2025 г. |
| Работу проверил  Кухар Д.А.\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc187256865)

[Алгоритм решения 4](#_Toc187256866)

[1. Ввод числа и проверка на корректный ввод. 5](#_Toc187256867)

[2. Вычисление внутреннего факториала. 5](#_Toc187256868)

[3. Вычисление факториала от факториала. 6](#_Toc187256869)

[Тестирование 8](#_Toc187256870)

[Тест 1: Проверка на наличие пользовательского интерфейса. 8](#_Toc187256871)

[Тест 2: Защита от некорректного ввода. 8](#_Toc187256872)

[Тест 3. Проверка корректности вычисления внутреннего факториала. 9](#_Toc187256873)

[Тест 4. Проверка корректности работы факториала от факториала. 9](#_Toc187256874)

[Код программы 11](#_Toc187256875)

# Постановка задачи

Вычислить точное значение (n!)! для n >= 3.

# Алгоритм решения

Данная выше задача была разбита на три этапа:

1. Ввод числа и проверка на корректный ввод.

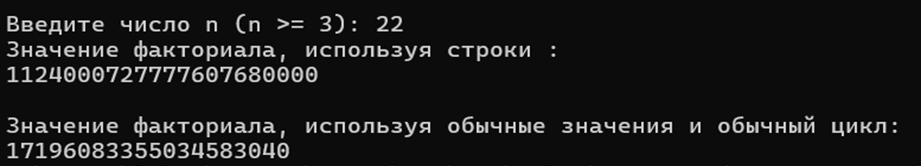
2. Вычисление внутреннего факториала в (n!)!

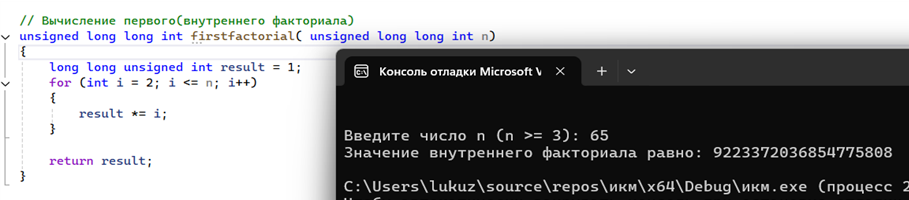
3. Вычисление факториала от факториала.

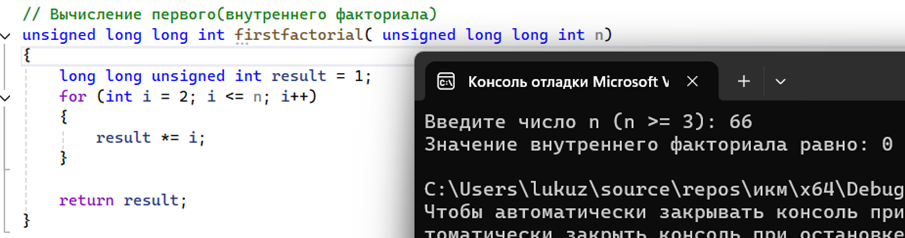
Впоследствии все этапы были реализованы.

Множество целых чисел, представимых в памяти ЭВМ, ограничено и определяется размером ячеек памяти (машинного слова), используемых для их хранения. В k-разрядной ячейке можно хранить 2k (655636) различных значений целых чисел. Этот факт имеет важное значение при выборе типа данных.

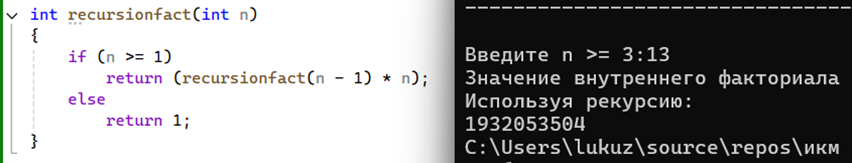
Также стоит заметить, что даже при использовании типа данных с максимальной емкостью (например, unsigned long long int, размером 8 байт) вычисление факториала остается сложной задачей. Обычные программы начинают выдавать неверные результаты уже при значениях факториала больше 20(см. источник 1). При попытке вычислить факториал числа 65 и выше результатом становится 0 (что объясняется переполнением). Например, для чисел 64 и 65 программа возвращает одно и то же значение — неверное в обоих случаях (см. источники 2, 3).

 (1).

 (2).

 (3).

Следует отметить, что факториал можно вычислять рекурсивно, однако при больших значениях возникает проблема переполнения памяти. Программа перестает корректно вычислять факториалы уже при значениях больше 12. (см. источник 4).

 (4).

В связи с этим было принято решение использовать строки для представления больших чисел и выполнения операций вычисления факториала. Такой подход позволяет избежать ограничений, накладываемых на объем представимых чисел при использовании стандартных типов данных.

## 1. Ввод числа и проверка на корректный ввод.

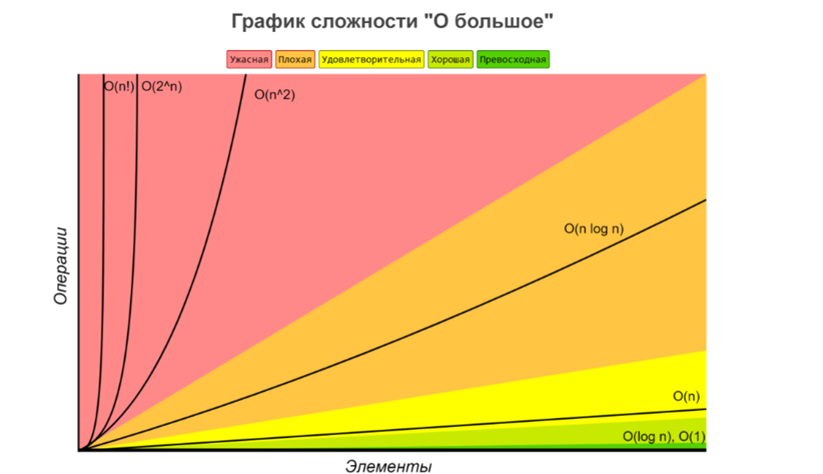
Для проверки корректности ввода числового значения используется отдельная функция isValidNumber, которая определяет, состоит ли весь введенный пользователем набор символов исключительно из цифр. Если в введённой строке обнаруживается хотя бы один символ, который не является цифрой от 0 до 9, программа выводит сообщение об ошибке и повторно запрашивает ввод, пока не будет получено корректное значение — целое число больше двух. После успешной проверки значения числа начинается второй этап — вычисление внутреннего факториала.

## 2. Вычисление внутреннего факториала.

Для расчета внутреннего факториала была выбрана структура данных вектор (динамический одномерный массив). Использование вектора оправдывается тем, что память для его элементов выделяется не на этапе компиляции, как в случае со статическими массивами, а во время выполнения программы. Это позволяет эффективно управлять памятью и упрощает обработку цифр по отдельности. В данном подходе каждая цифра числа записывается в векторе в обратном порядке (начиная с младшего разряда). Затем функция multiply выполняет поочередное умножение текущего результата на новое значение, полученное на предыдущих итерациях. После завершения всех операций итоговое число приводится к стандартному виду (путем переворота порядка цифр) и выводится на экран для удобства пользователя.

## 3. Вычисление факториала от факториала.

Заключительный и наиболее сложный этап работы — вычисление факториала от факториала. Следует отметить, что факториал имеет самый быстрый темп роста по сравнению с другими функциями (см. источник 5). Это делает задачу особенно сложной и требует оптимального подхода к обработке данных.

 (5).

Для оптимизации вычисления факториала от факториала был разработан алгоритм, который снижает вычислительную сложность при увеличении входных данных, стремящихся к бесконечности. В основе программы лежит метод Карацубы — эффективный алгоритм быстрого умножения чисел.

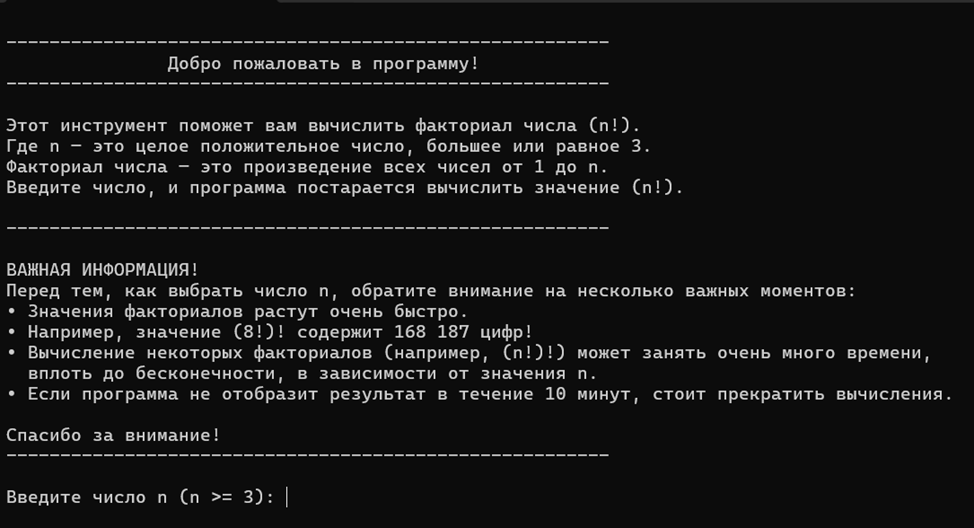
Суть этого метода заключается в разбиении каждого числа на части и поочередном перемножении всех разрядов одного числа на разряды другого. Это позволяет значительно ускорить вычисления по сравнению с обычным подходом, где производится последовательное умножение.

Для реализации этого метода каждое число разделяется на блоки, которые затем перемножаются между собой с использованием функций multiplyLargeNumber и factorialKaratsuba. В этих функциях, как и ранее, применяется вектор для хранения цифр числа, что позволяет удобно обрабатывать каждый символ по отдельности. Такой подход обеспечивает эффективность и надежность выполнения алгоритма даже для больших объемов вычислений.

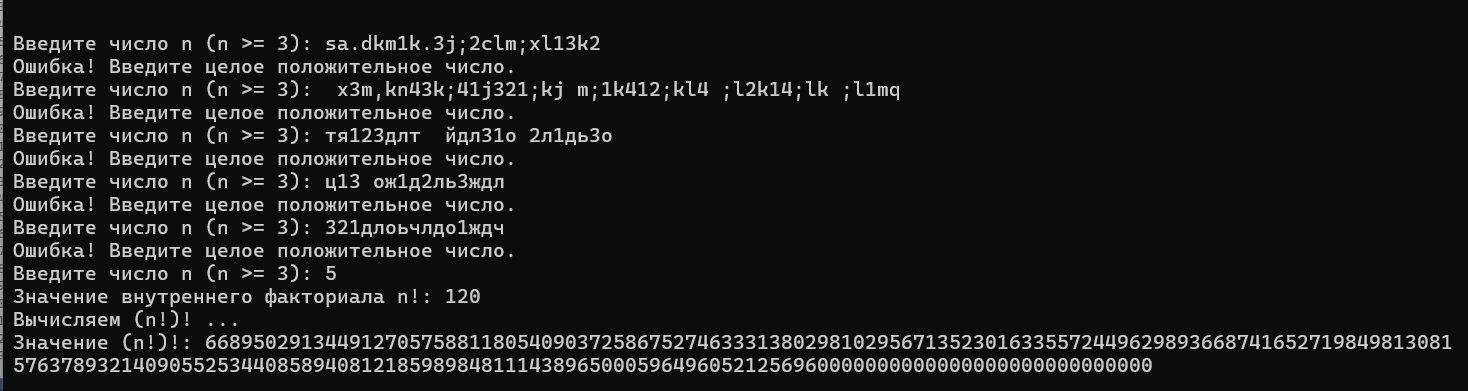
Таким образом, общая сложность работы алгоритма находится между O (n2\* ) и O (n2.585\*2). Асимптотическая сложность предоставленного алгоритма выше, чем у стандартного метода. Однако на практике этот алгоритм выигрывает при вычислениях с очень большими числами. Это обусловлено использованием метода Карацубы, который значительно ускоряет умножение больших чисел, особенно когда их длина становится очень большой. Таким образом, несмотря на высокую асимптотическую сложность, алгоритм эффективнее для обработки реальных данных с большими объемами вычислений.

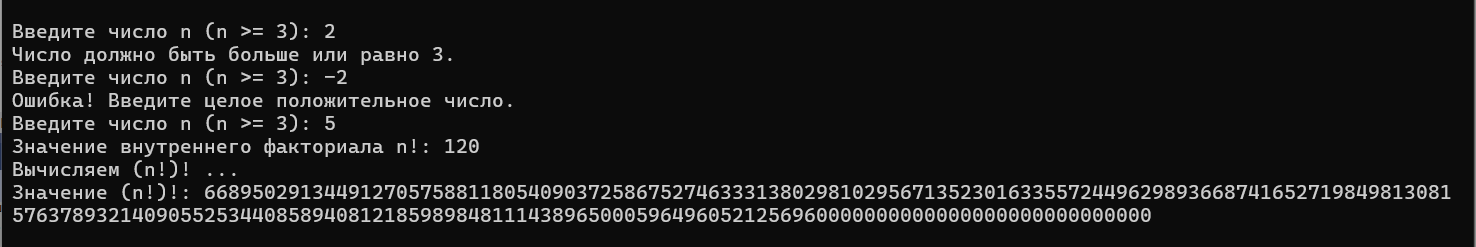
# Тестирование

## Тест 1: Проверка на наличие пользовательского интерфейса.

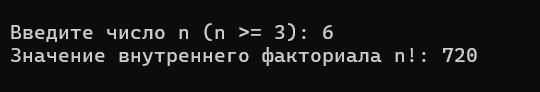
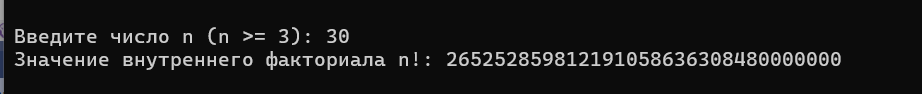
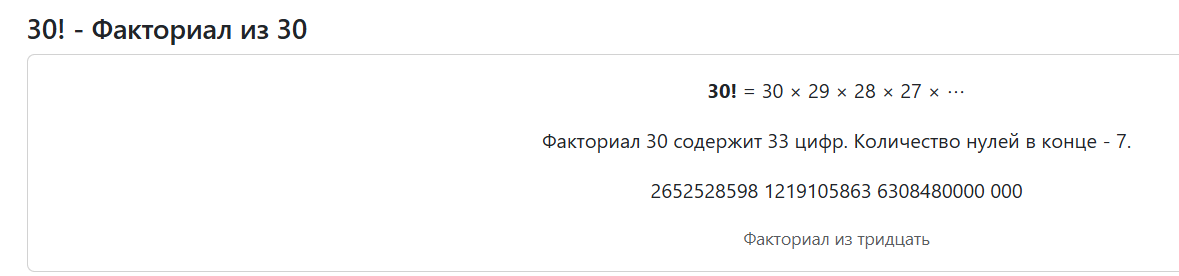


## Тест 2: Защита от некорректного ввода.



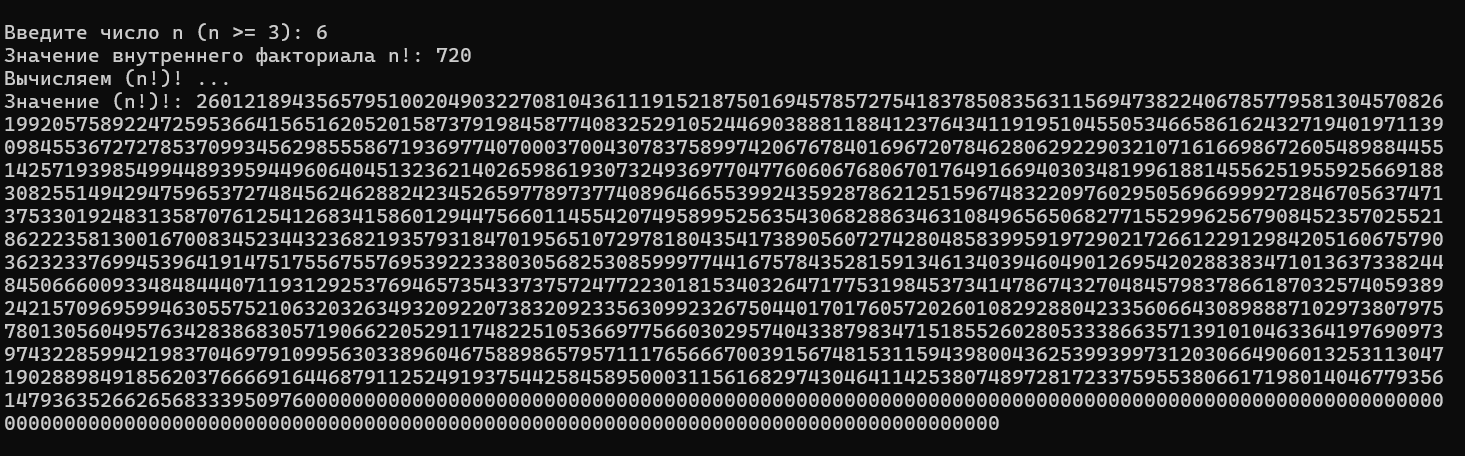


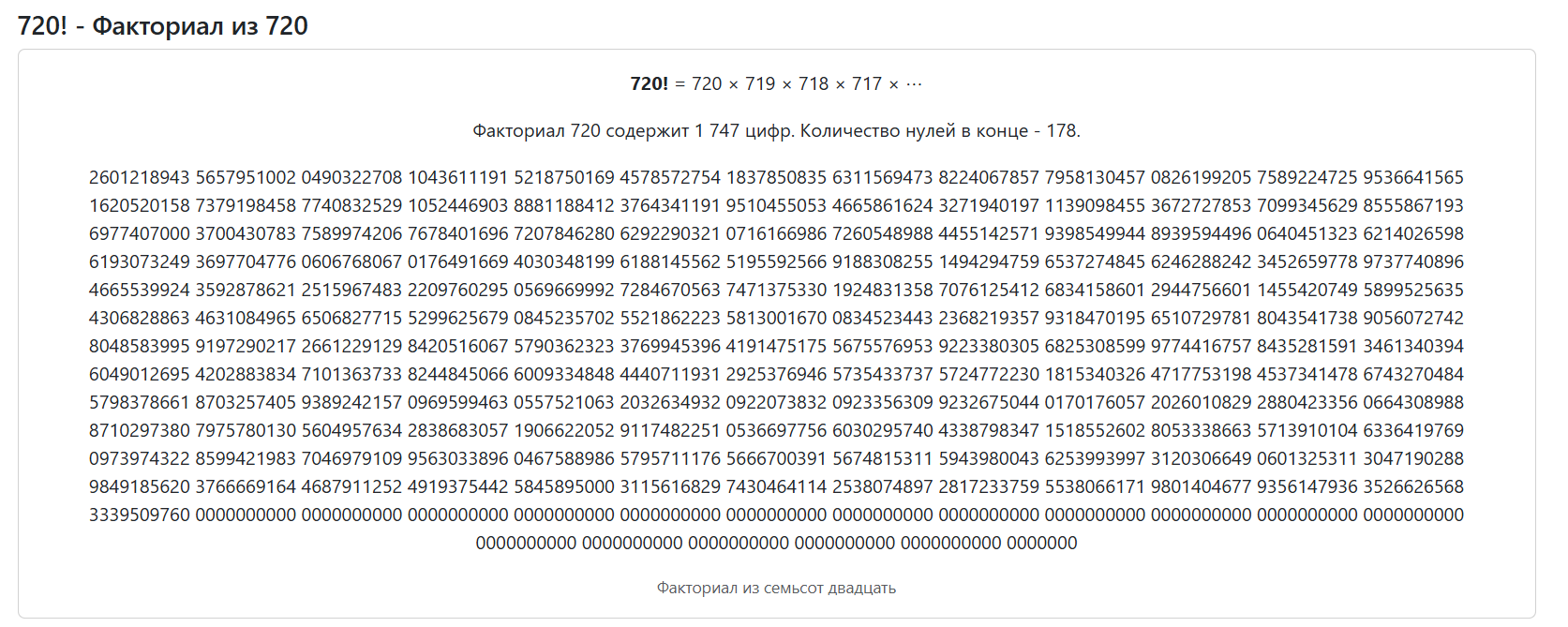
## Тест 3. Проверка корректности вычисления внутреннего факториала.

- Проверено на больших числах.

## Тест 4. Проверка корректности работы алгоритма факториала от факториала.

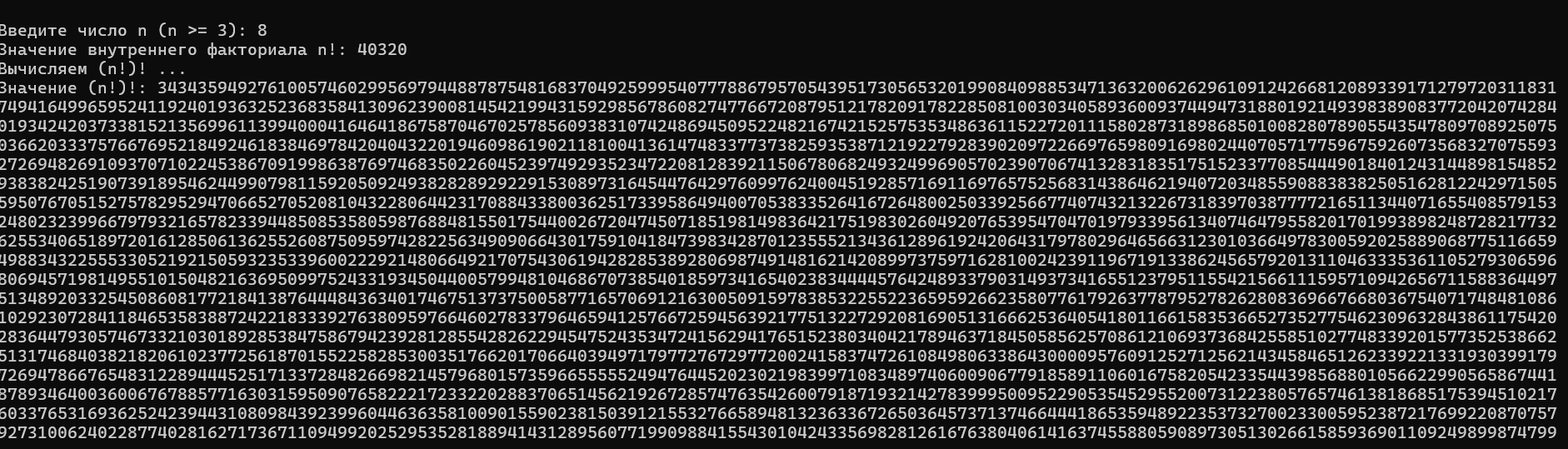




(9). Значение, найденное в интернете.

Для значений больше 7 время работы программы резко возрастает, несмотря на оптимизацию алгоритма. Видео – полный результат вычисления (8!)! или (40320)!.

<https://disk.yandex.ru/i/nRPraQgC2ObTOQ>

<https://zeptomath.com/calculators/factorial.php?number=40320> – Проверка на корректность.

(Часть ответа, посчитанного программой).

# Код программы

<https://github.com/Kuzhlevait17/task-18>

<https://github.com/Kuzhlevait17/task-18.git>