МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ФИТ

Дисциплина Математическое программирование

**Отчет по лабораторным работам по дисциплине**

**“Математическое программирование”**

Выполнил: студент 2курса 4 группы Цывинская А.Е.

(Ф.И.О)

Минск 2025

**Лабораторная работа №1**

Вспомогательные функции

**Задание 1**

Разработайте три функции (start, dget и iget), используя следующие спецификации (рисунок 1):

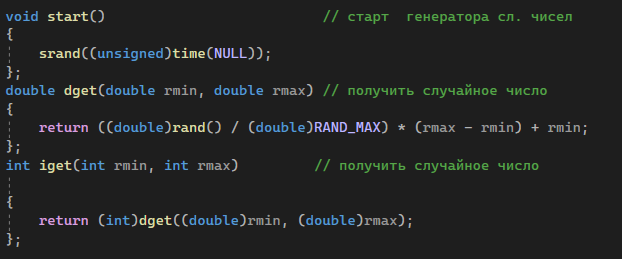


Рисунок 1 – Реализация функций

**Задание 2**

1.Реализовать пример 2.

2.Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

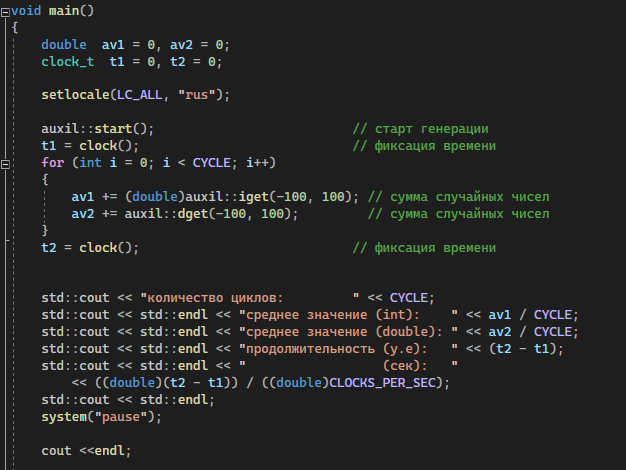


Рисунок 2 – вызов функции и измерение времени выполнения

**Задание 3**

Проведите необходимые эксперименты (разработать кодом) и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2.

Примечание: продолжительность вычисления измерять в условных единицах процессорного времени (функция clock). График представлен на рисунке 3

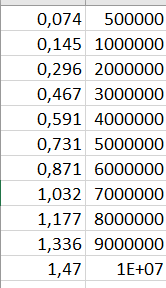
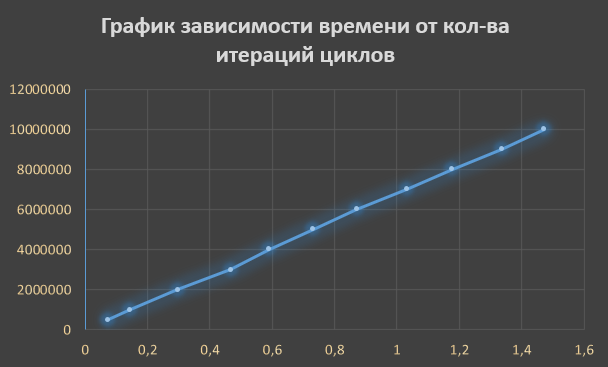


Рисунок 3 – график зависимости

Реализация Факториала в коде демонстрируется на рис. 4

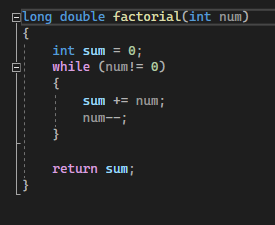


Рисунок 4 – функция для нахождения факториала

Вызов и подсчет времени для реализации факториала представлен на рисунке 5:

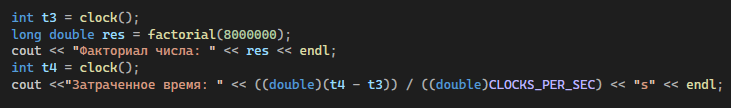


Рисунок 5 – вызов функции для подсчёта факториала

График зависимости:

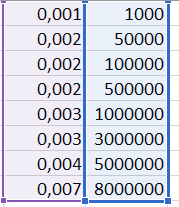
 

Рисунок 6 – график зависимости

**Вывод**: по полученным измерениям, можем сделать следующий вывод, что время выполнения программы линейно зависит от количества циклов (Значения факториала).

**Лабораторная работа №2**

Комбинаторные алгоритмы решения оптимизационных задач

**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, **перестановок, сочетаний и размещений** на С++; научиться применять разработанные генераторы для решения задач о рюкзаке (упрощенную, коммивояжера, об оптимальной загрузке судна и об оптимальной загрузке судна с центровкой.

**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества.

**Описание процесса работы генератора подмножеств:**

Используется битовая маска **(**у нас есть множество из n элементов и мы перебираем все числа от 0 до 2^n - 1, где каждый бит указывает, включён ли элемент в подмножество)

Есть еще основные методы класса subset, они представлены на рисунке 1.

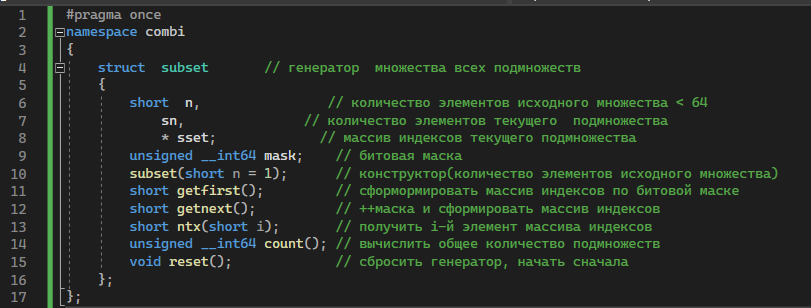


Рисунок 1 – Combi.h

Сombi.cpp реализует методы генератора подмножеств  
#include "Combi.h"

#include <algorithm>

#include <iostream>

namespace combi

{

subset::subset(short n) //конструктор

{

this->n = n; //сохранение размеры множества

this->sset = new short[n]; //выделение памяти под массив индексов подмножества

this->reset(); //сброс

};

void subset::reset()

{

this->sn = 0; //размер текущего подмножества (изначально пустое)

this->mask = 0; //обнуляем битовую маску чтобы начать с пустого множества

};

short subset::getfirst() //подмножество будем формировать из битовой маски

{

\_\_int64 buf = this->mask; //копирование маски

this->sn = 0; //сброс размера текущего подмножества

for (short i = 0; i < n; i++) // по битам

{

if (buf & 0x1) // если младший бит 1

this->sset[this->sn++] = i; //добавялем индекс в sset

buf >>= 1; // сдвиг вправо и проверка некст бита

}

return this->sn;

};

// Копируем mask в buf, затем идем по всем битам buf, если i-й бит 1, добавляем i в sset, в конце sn содержит размер подмножества

short subset::getnext() // для получения след подмножества

{

int rc = -1; // возвращаемое значение -1

this->sn = 0; //сброс размера подмножества

if (++this->mask < this->count()) //++ маску и проверка на лимит

rc = getfirst(); // нью подмножество

return rc;

};

// Увеличиваем mask++, чтобы получить следующее подмножество; если mask не превысило 2^n, вызываем getfirst() для формирования нового подмножества; если все подмножества перебраны, возвращает -1

short subset::ntx(short i)

{

return this->sset[i]; // итый элемент текущ подмножества (ntx в мейне будет использоватсья чтобы вывести уже в последствии все элементы)

};

unsigned \_\_int64 subset::count() //кол-во подмножеств

{

return (unsigned \_\_int64)(1 << this->n); // 2^n подмножеств

};

};

В main\_ex1.cpp создаётся генератор подмножеств и перебираются все возможные подмножества множества {A, B, C, D}

#include <iostream>

#include "Combi.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

char AA[][2] = { "A", "B", "C", "D" }; //инициализируем множества (массив строк литералов)

std::cout << std::endl << " - Генератор множества всех подмножеств -";

std::cout << std::endl << "Исходное множество: ";

std::cout << "{ ";

for (int i = 0; i < sizeof(AA) / 2; i++)

std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " "); //вычисление кол-ва элементов АА (размер АА в байтах делится на размер одного элемента)

std::cout << "}";

std::cout << std::endl << "Генерация всех подмножеств ";

combi::subset s1(sizeof(AA) / 2); // создание генератора (**Создаётся объект s1 класса combi::subset**)

int n = s1.getfirst(); // первое (пустое) подмножество

while (n >= 0) // пока есть подмножества

{

std::cout << std::endl << "{ ";

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

std::cout << AA[s1.ntx(i)] << ((i < n - 1) ? ", " : " ");

//Проходим по массиву индексов sset[], который хранит текущее подмножество; s1.ntx(i) возвращает индекс элемента множества (например, 0 → "A", 1 → "B" и т. д.); Выводим текущее подмножество в формате { A, B }

std::cout << "}";

n = s1.getnext(); // cледующее подмножество

};

std::cout << std::endl << "всего: " << s1.count() << std::endl;

system("pause");

return 0;

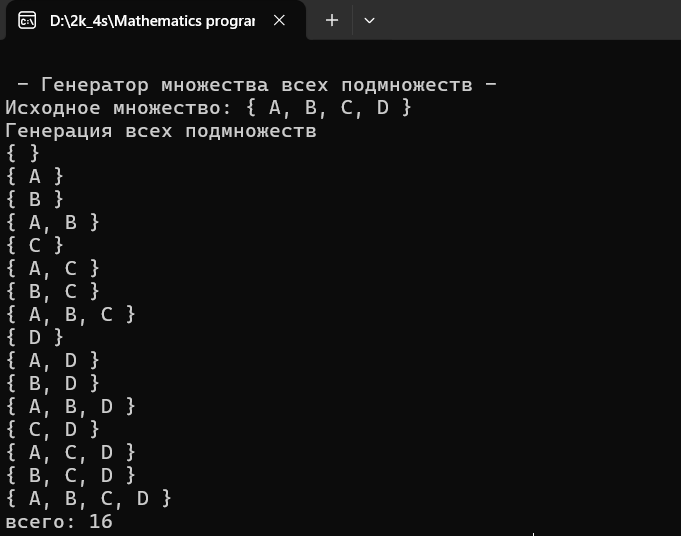
}

Битовая маска перебирает все комбинации от 0000₂ до 1111₂

Каждый бит указывает, включён ли элемент множества

Все подмножества выводятся последовательно

В конце программа выводит 2⁴ = 16 т е общее число подмножеств

Рисунок – Результат выполнения

**Задание 2.** Разобрать и разработать генератор сочетаний.

В этом задании было необходимо разработать генератор сочетаний. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

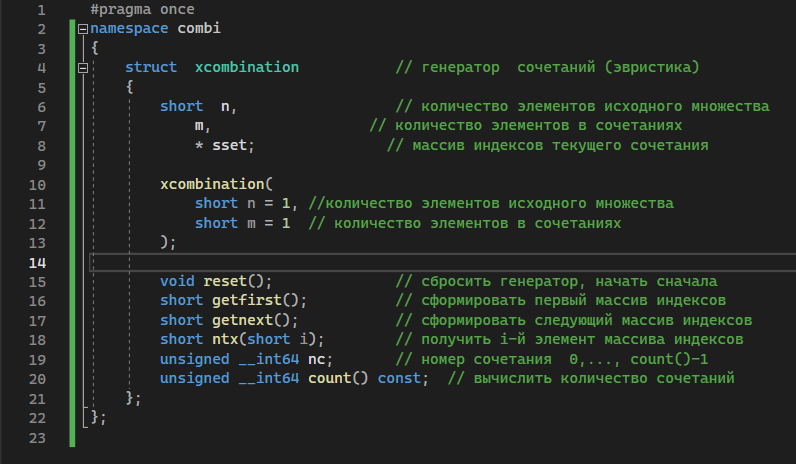


Рисунок 5 - Combi.h

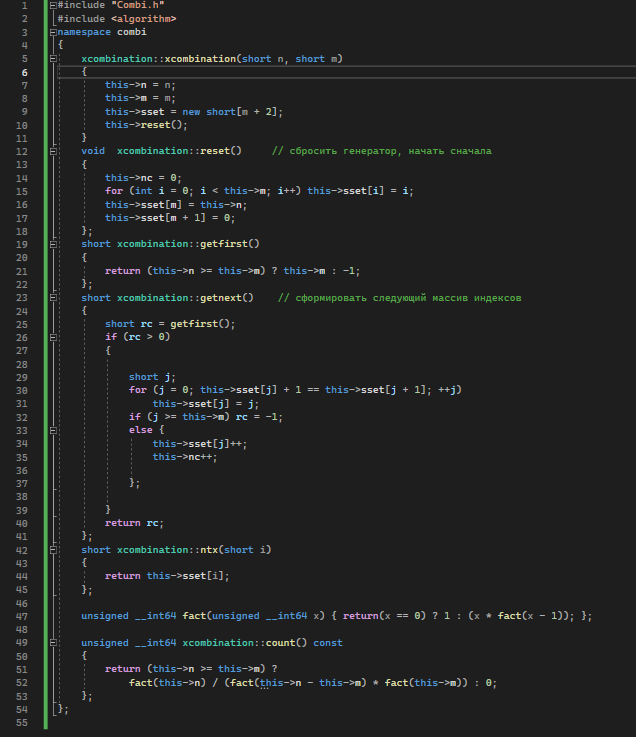


Рисунок 6 - Combi.cpp

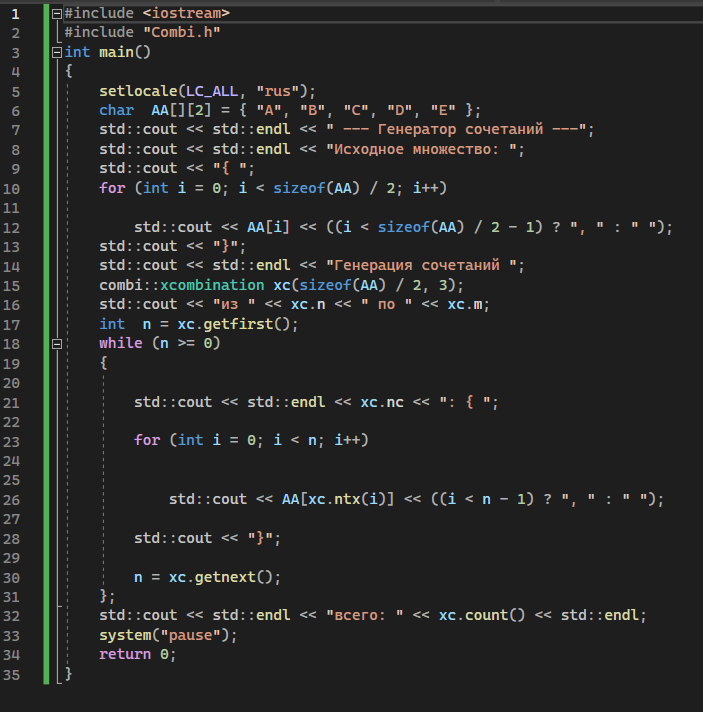


Рисунок 7 - Main\_ex2.cpp

Результат выполнения представлен на рисунке 8.

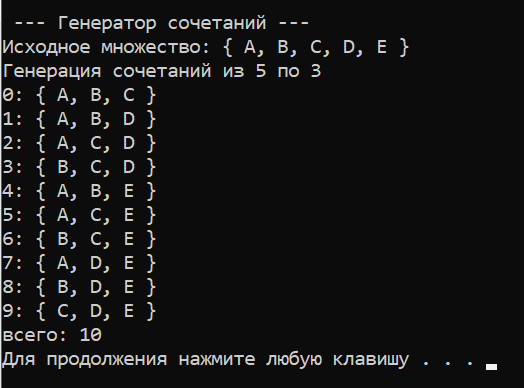


Рисунок 8 – результат выполнения программы

**Задание 3.** Разобрать и разработать генератор перестановок.

В этом задании было необходимо разработать генератор перестановок. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

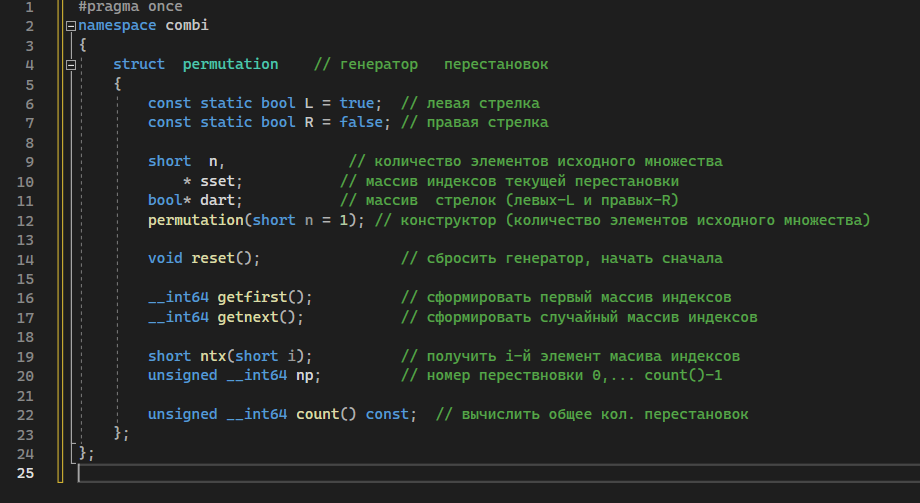


Рисунок 9 – Combi.h

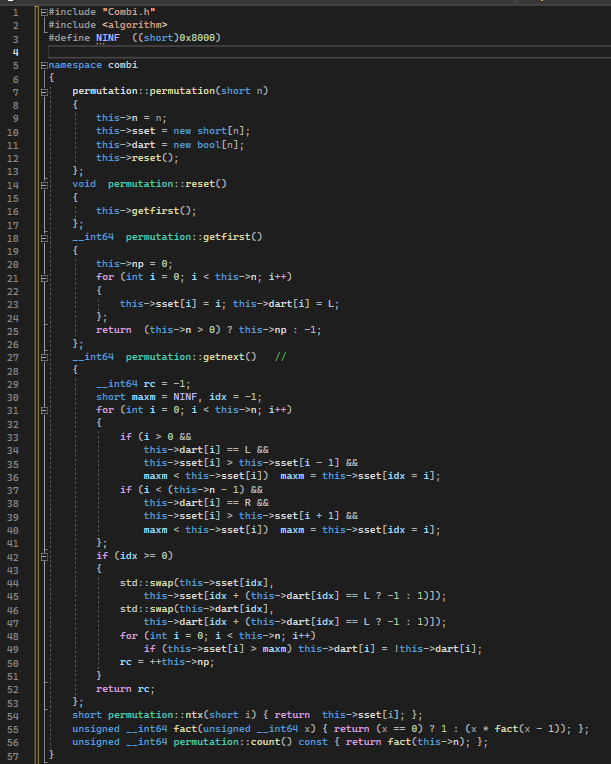


Рисунок 10 – Combi.cpp

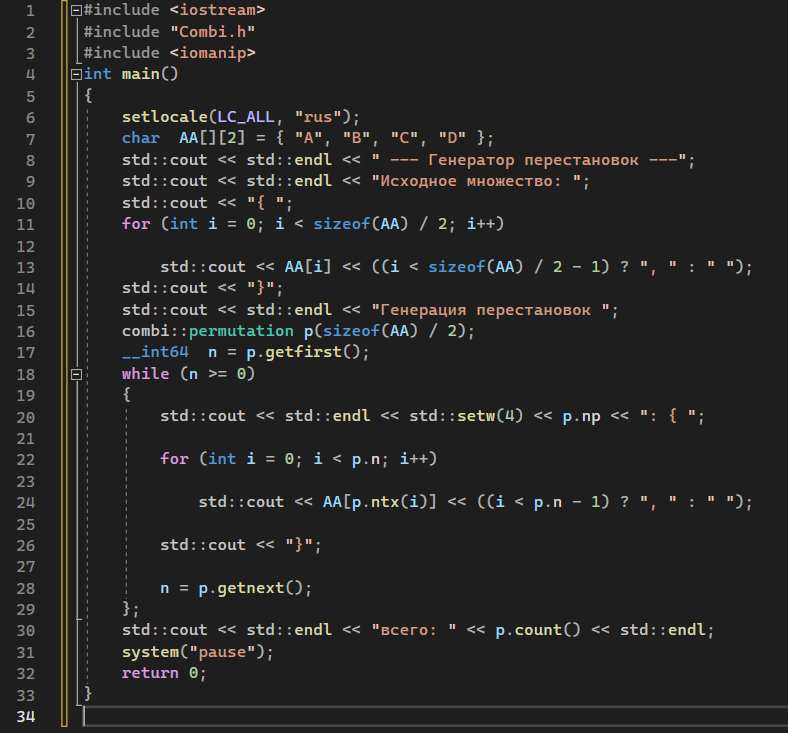


Рисунок 11 – Main\_ex3.cpp

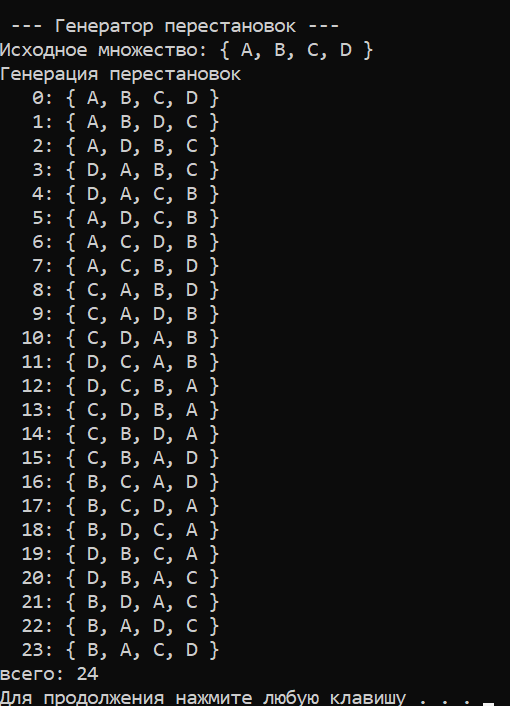


Рисунок 12 – результат выполнения программы

**Задание 4.** Разобрать и разработать генератор размещений.

В этом задании было необходимо разработать генератор размещений. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

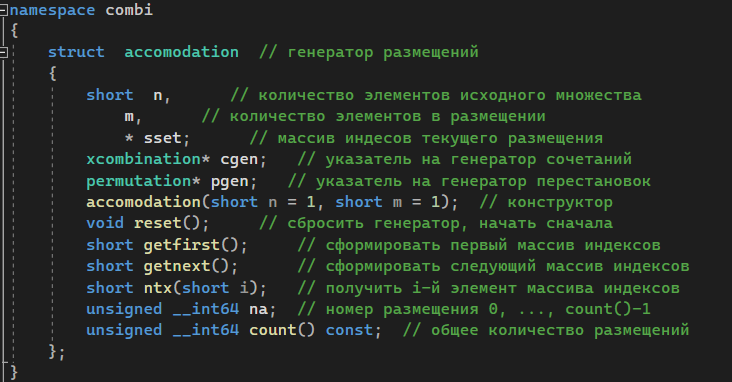


Рисунок 13 – Combi.h



Рисунок 14 – Combi.cpp

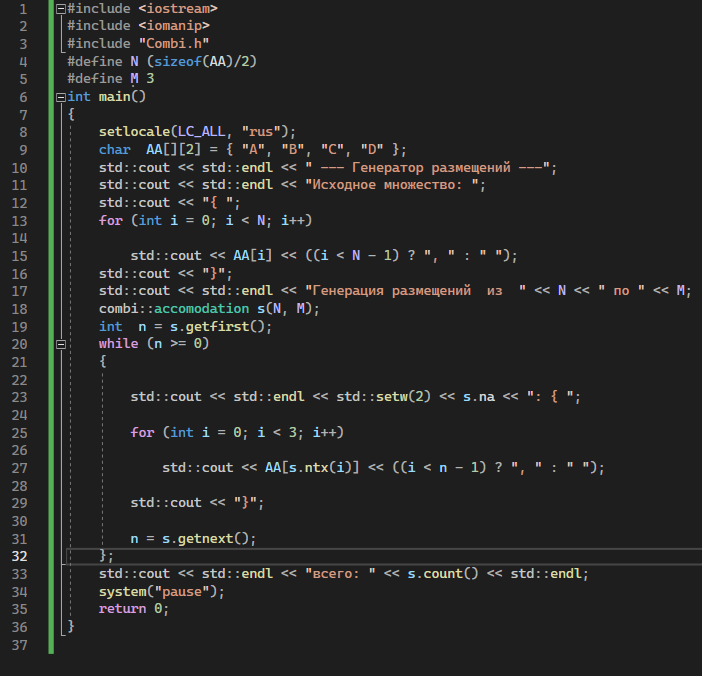


Рисунок 15 – Main\_ex4.cpp

Результат выполнения программы представлен на рисунке 16.

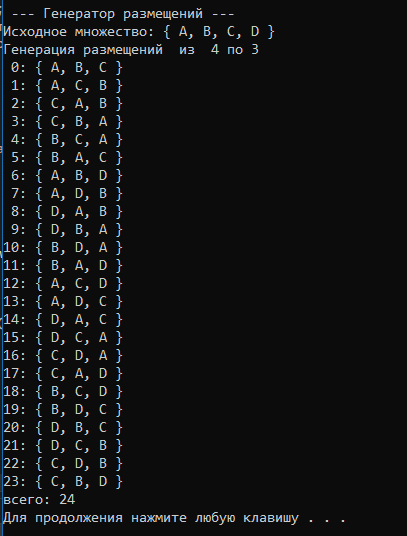


Рисунок 16 - результат выполнения программы

**Задание 5.** Решить в соответствии с вариантом задачу и результат занести в отчет. У меня 7ой вариант, поэтому условие задачи следующее:

об оптимальной загрузке судна (веса контейнеров сгенерировать случайным образом: ограничение по общему весу – 1500 кг., количество мест на судне для контейнеров – 5, количество контейнеров 25, веса контейнеров 100 – 900 кг., доход от перевозки 10 – 150 у.е.);

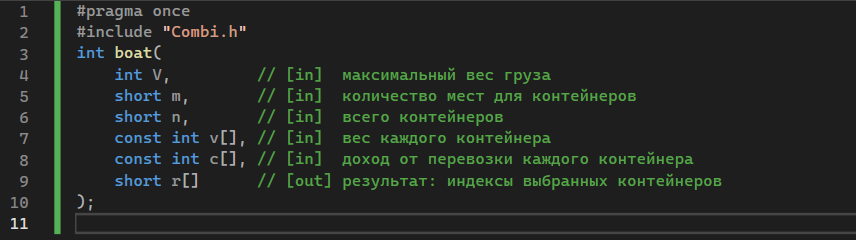


Рисунок 17 – Boat.h

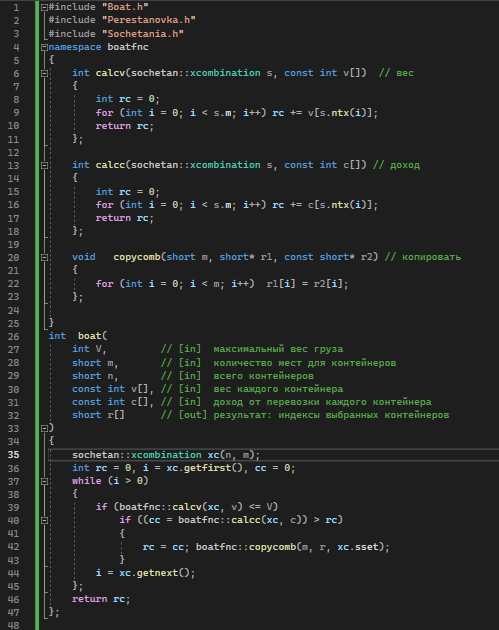


Рисунок 18 – Boat.cpp

Результат выполнения программы представлен на рисунке 19.

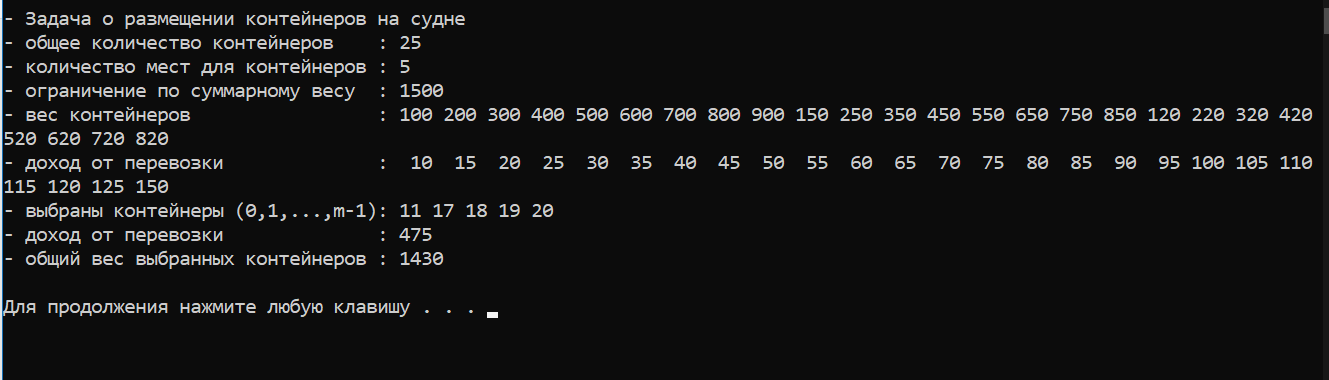


Рисунок 19 – результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи от размерности задачи и результат в виде графика:

об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество контейнеров 25 – 35

Результат работы программы на рисунке 20.

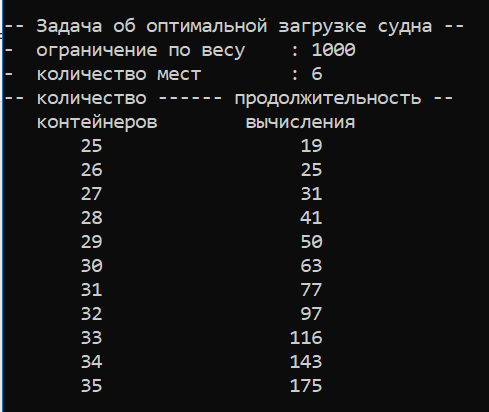


Рисунок 20 – Зависимость времени выполнения от количества контейнеров



Рисунок 21 – график зависимости

Вывод: исходя из полученных данных и графика, можно заметить, что скорость выполнения программы плавно возрастает при добавлении количества контейров.