МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ФИТ

Дисциплина Математическое программирование

**Отчет по лабораторным работам по дисциплине**

**“Математическое программирование”**

Выполнил: студентки 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):

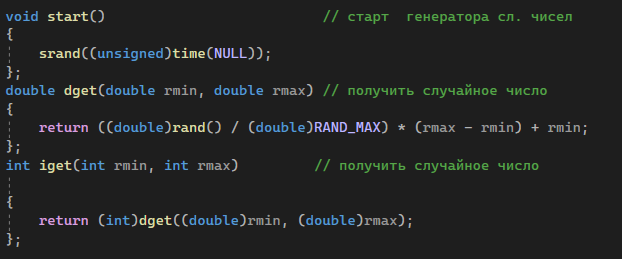


Рисунок 1 – Реализация функций

**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.

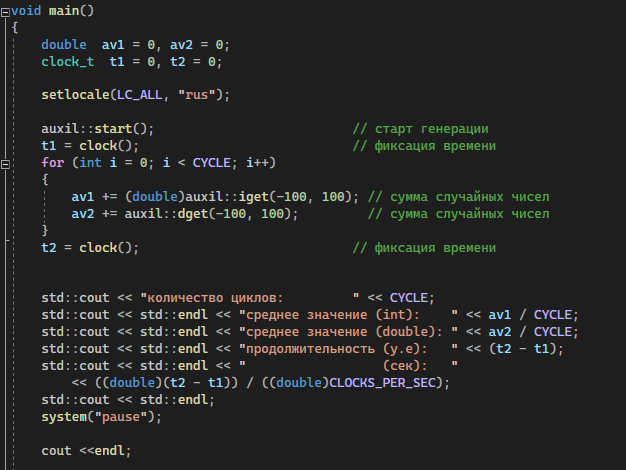


Рисунок 2 – вызов функции и измерение времени выполнения

**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

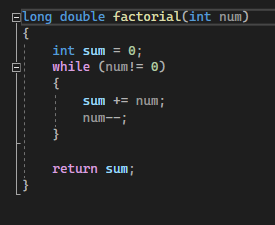


Рисунок 4 – функция для нахождения факториала

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

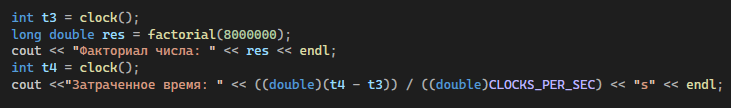


Рисунок 5 – вызов функции для подсчёта факториала

**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.

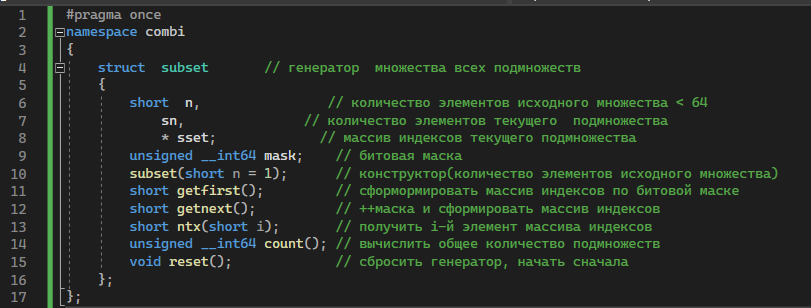


Рисунок 1 – Combi.h

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;

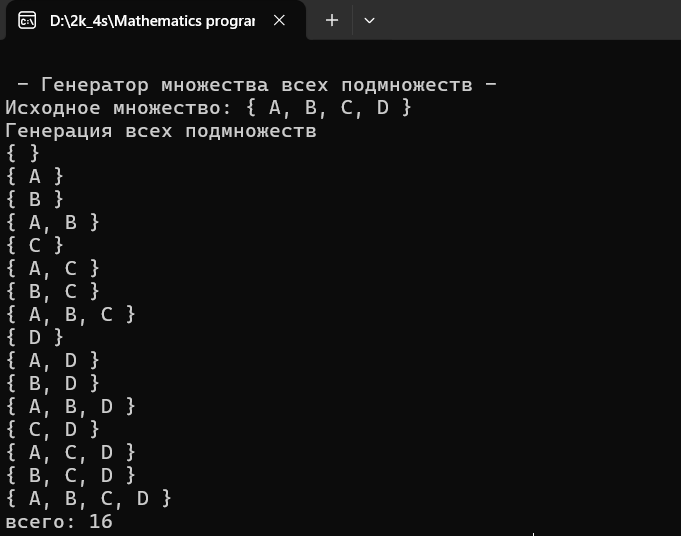
}

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.

Программа будет реализовывать генератор сочетаний (комбинаций) т е генерировать всевозможные сочетания фиксированного размера m из множества из n элементов.

Например, если у нас есть множество {A, B, C, D, E} и мы хотим получить все сочетания по 3 элемента, программа выдаст:

* {A, B, C}
* {A, B, D}
* {A, B, E}
* ...
* {C, D, E}

На рисунке 3 представлен заголовочный файл

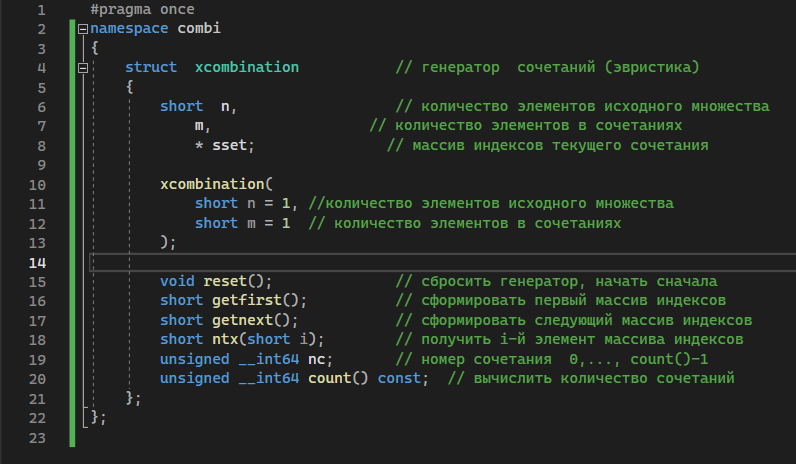


Рисунок 3 - Combi.h

**xcombination** — это структура (почти как класс, но без методов private и protected).

n — общее количество элементов в исходном множестве.

m — количество элементов в сочетаниях.

sset — массив индексов текущего сочетания.

xcombination(short n = 1, short m = 1); // Конструктор, создающий генератор сочетаний из n элементов по m.

reset() — сбрасывает генератор (возвращает его в начальное состояние).

getfirst() — создаёт первый набор индексов.

getnext() — создаёт следующий набор индексов.

ntx(i) — возвращает i-й индекс текущего сочетания.

nc — номер текущего сочетания (от 0 до count() - 1).

count() — вычисляет общее количество сочетаний.

Дальше разьерем Combi.cpp

#include "Combi.h"

#include <algorithm>

namespace combi

{

xcombination::xcombination(short n, short m)

{

this->n = n;

this->m = m;

this->sset = new short[m + 2];

this->reset();

}

//конструктор который создает объект икскомбинейшн с заданными n и m; sset – массів длиной m+2 и вызывается ресет для сброса генератора

void xcombination::reset() // сбросить генератор, начать сначала

{

this->nc = 0; //начиная с первого сочетания

for (int i = 0; i < this->m; i++) this->sset[i] = i; //проход по массиву и заполнение массива индексами первого сочетания

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

short xcombination::getfirst()

{

return (this->n >= this->m) ? this->m : -1; // можем вообще такое сочетание создать? Да - m

};

short xcombination::getnext() // сформировать следующий массив индексов

{

short rc = getfirst();

if (rc > 0)

{

short j;

for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)

this->sset[j] = j;

if (j >= this->m) rc = -1;

else {

this->sset[j]++;

this->nc++;

};

}

return rc;

};

//тут мы проверяли есть ли еще сочетания и сдвигали самый правый индекс если это возможно а если все индексы достигли предела то конец программы

short xcombination::ntx(short i) //получение i-го элемента текущего сочетания

{

return this->sset[i];

};

unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); }; //РЕКУРСИЯ ДЛЯ ФАКТОРИАЛА

unsigned \_\_int64 xcombination::count() const

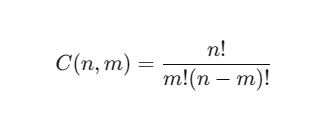
{

return (this->n >= this->m) ?

fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;

};

};



Файл main\_ex2.cpp представлен на рисунке 4.

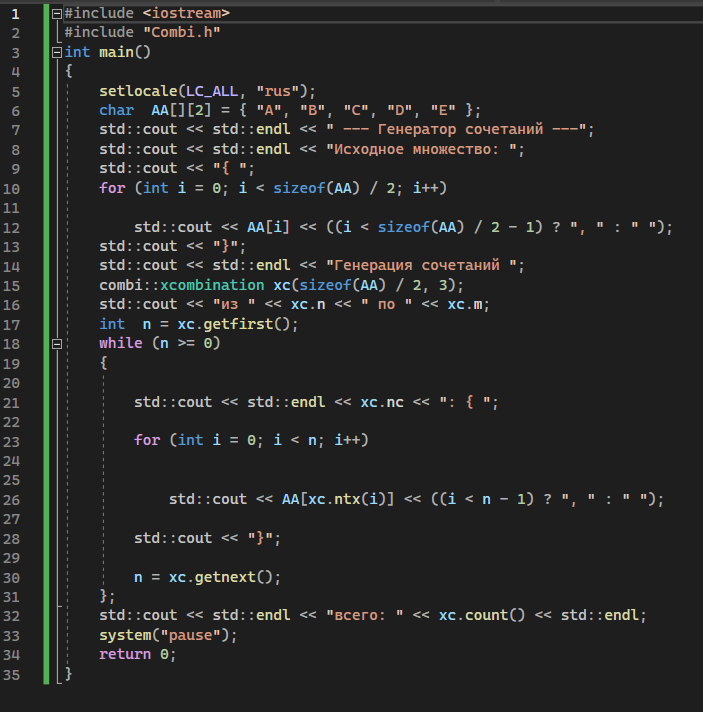


Рисунок 4 - Main\_ex2.cpp

combi::xcombination xc(sizeof(AA) / 2, 3);

Создаём генератор сочетаний из 5 элементов по 3

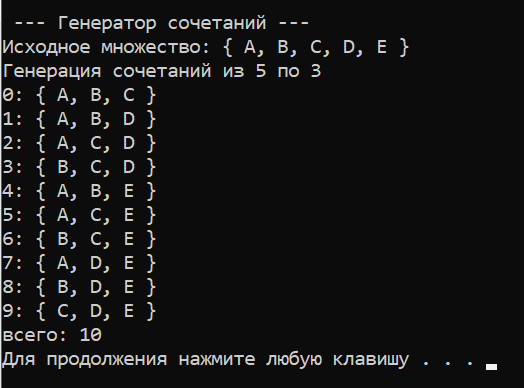
Всего C(5,3) = 10 сочетаний

Рисунок 5 – Результат выполнения программы

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

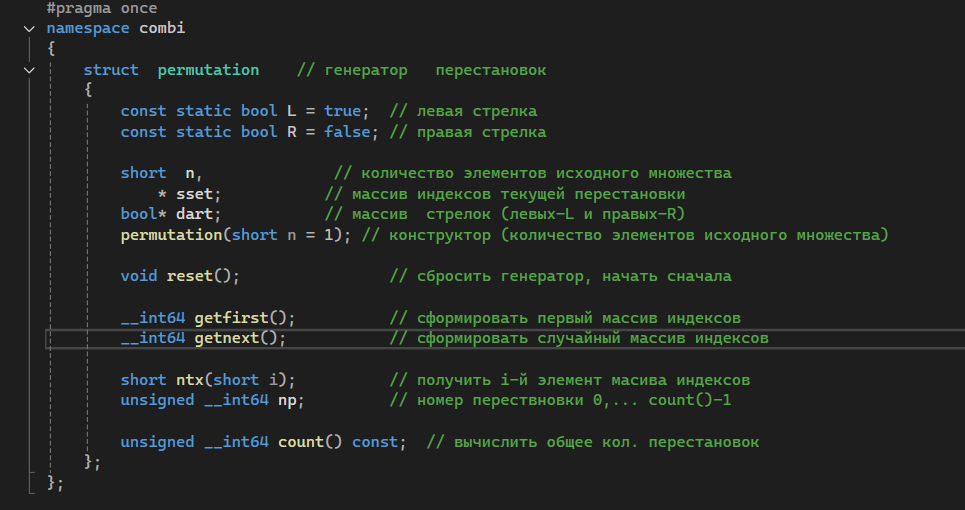


Рисунок 6 – Combi.h

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

permutation::permutation(short n) //реализация класса перестановок

{

this->n = n;

this->sset = new short[n]; //выделение места

this->dart = new bool[n];

this->reset(); // для установки начального состояния

};

void permutation::reset()

{

this->getfirst(); //вызов гетферста чтобы сбросить генератор и начать заново

};

\_\_int64 permutation::getfirst() // установка первой перестановки

{

this->np = 0;

for (int i = 0; i < this->n; i++)

{

this->sset[i] = i;

this->dart[i] = L;//направление влево

};

return (this->n > 0) ? this->np : -1;

};

\_\_int64 permutation::getnext() //генерация следующей перестановки

{

\_\_int64 rc = -1; //если перестановки кончились то -1 вернется

short maxm = NINF,// хранит максимально подвижное число

idx = -1; //индекс числа

for (int i = 0; i < this->n; i++) //поиск наибольшего подвижного числа

{

if (i > 0 &&

this->dart[i] == L &&

this->sset[i] > this->sset[i - 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

if (i < (this->n - 1) &&

this->dart[i] == R &&

this->sset[i] > this->sset[i + 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

};

if (idx >= 0)//перемещение максимального подвижного числа

{

std::swap(this->sset[idx],

this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);

std::swap(this->dart[idx],

this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);

for (int i = 0; i < this->n; i++) // Все числа **больше** maxm **меняют направление**

if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];

rc = ++this->np; //обновление номера перестановки

}

return rc;

};

short permutation::ntx(short i) { return this->sset[i]; };

unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };

unsigned \_\_int64 permutation::count() const { return fact(this->n); };

}

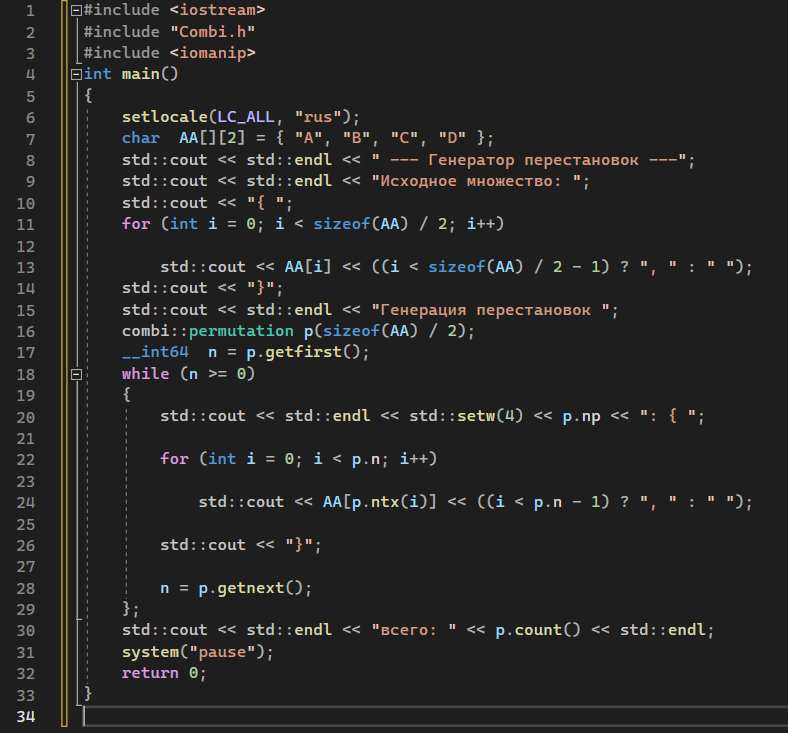


Рисунок 7 – Main\_ex3.cpp

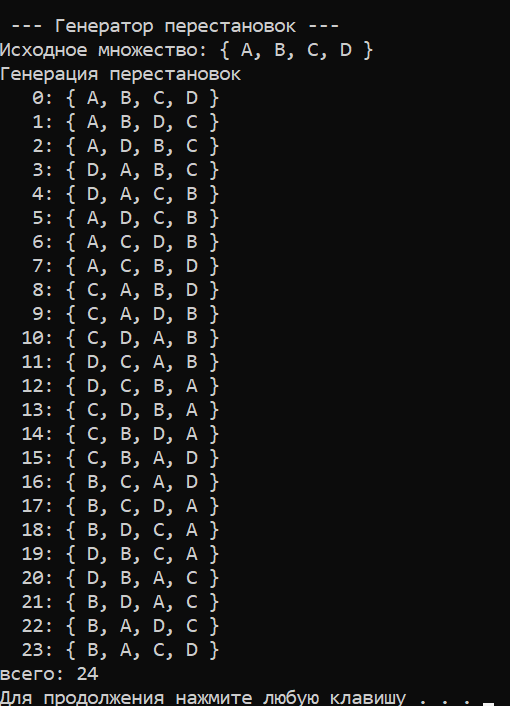


Рисунок 8 – Результат выполнения программы

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

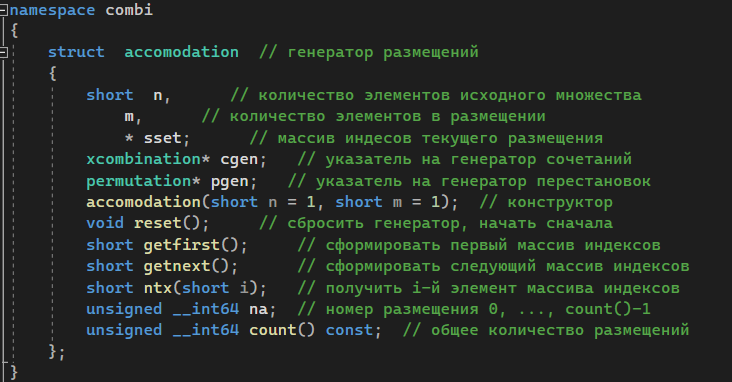


Рисунок 9 – Combi.h



Рисунок 10 – Combi.cpp

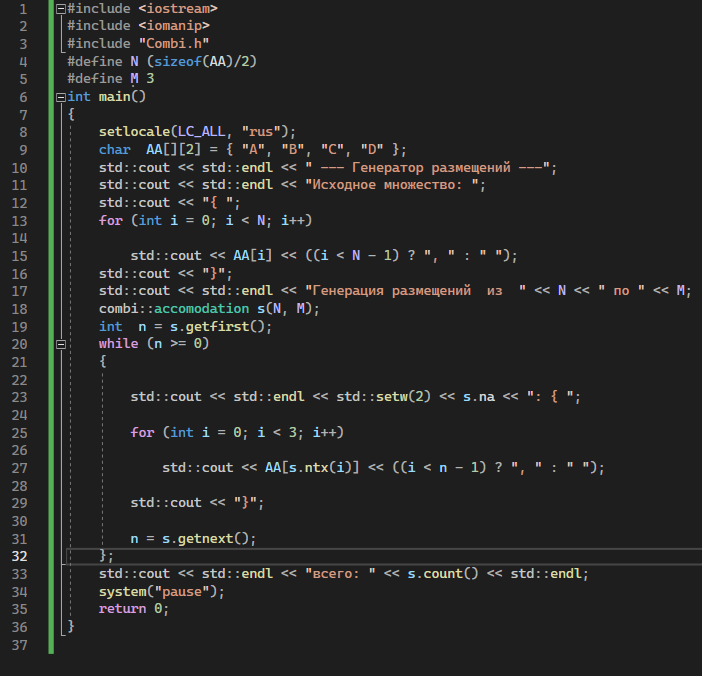


Рисунок 11 – Main\_ex4.cpp

Результат выполнения программы представлен на рисунке 12

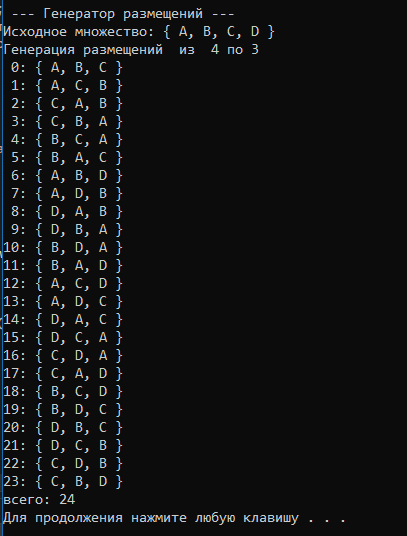


Рисунок 12 - результат выполнения программы

**Задание 5.** Решить в соответствии с вариантом задачу и результат занести в отчет. У меня 12-ый вариант, поэтому условие задачи следующее:

об оптимальной загрузке судна с условием центровки (веса контейнеров сгенерировать случайным образом: количество мест на судне для контейнеров – 5, количество контейнеров 8, веса контейнеров 100 – 200 кг., доход от перевозки 10 – 100 у.е.; минимальный вес контейнера для каждого места 50 – 120 кг, максимальный вес контейнера для каждого места 150 – 850 кг);

Алгоритм решения

1. Генерация случайных данных:
   * Генерируем 8 контейнеров, у каждого случайный вес (100-200 кг) и доход (10-100 у.е.)
   * Генерируем 5 мест на судне с ограничениями на мин. (50-120 кг) и макс. (150-850 кг) допустимый вес контейнера
2. Оптимизация загрузки:
   * Контейнеры сортируются по коэффициенту прибыльности (profit weight), чтобы сначала загружать наиболее выгодные
   * Контейнер загружается в первое подходящее место, где он удовлетворяет условиям веса
   * Если место занято или контейнер не подходит по весу, он остается незагруженным
3. Вывод результата:
   * Показываем, какие контейнеры загружены и в какие места
   * Вычисляем и выводим общий доход от перевозки

**Сombi.h**

#pragma once

namespace combi {

struct ship\_loading { // Структура для управления загрузкой судна

struct Container { // Описание контейнера

int id; // Номер контейнера

int weight; // Вес контейнера

int profit; // Доход от перевозки

};

struct Place { // Описание места на судне

int min\_weight; // Минимальный допустимый вес контейнера

int max\_weight; // Максимальный допустимый вес контейнера

bool occupied; // Флаг, занято ли место

};

Container containers[8]; // Массив контейнеров

Place places[5]; // Массив мест

int loaded[8]; // В каком месте находится контейнер

ship\_loading(); // Конструктор (инициализация)

void generate\_data(); // Генерация случайных данных

void optimize\_loading(); // Оптимальная загрузка

void print\_result(); // Вывод результата

};

}

**Combi.cpp**

#include "Combi.h"

#include <iostream>

#include <algorithm>

#include <ctime>

namespace combi {

ship\_loading::ship\_loading() {

for (int i = 0; i < 8; i++) loaded[i] = -1; // Изначально контейнеры не размещены

generate\_data();

}

void ship\_loading::generate\_data() {

srand(time(0)); // Инициализация генератора случайных чисел

for (int i = 0; i < 8; i++) { // Заполняем контейнеры

containers[i].id = i + 1;

containers[i].weight = rand() % 101 + 100; // Вес 100-200 кг

containers[i].profit = rand() % 91 + 10; // Доход 10-100 у.е

}

for (int i = 0; i < 5; i++) { // Заполняем места на судне

places[i].min\_weight = rand() % 71 + 50; // Мин. вес 50-120 кг

places[i].max\_weight = rand() % 701 + 150; // Макс. вес 150-850 кг

places[i].occupied = false; // Все места изначально свободны

}

}

void ship\_loading::optimize\_loading() {

// Сортируем контейнеры по прибыльности (доход вес)

std::sort(containers, containers + 8, [](const Container& a, const Container& b) {

return (double)a.profit / a.weight > (double)b.profit / b.weight;

});

for (int i = 0; i < 8; i++) { // Перебираем контейнеры

for (int j = 0; j < 5; j++) { // Перебираем места на судне

if (!places[j].occupied && // Если место свободно

containers[i].weight >= places[j].min\_weight && // Вес контейнера подходит

containers[i].weight <= places[j].max\_weight) {

loaded[i] = j; // Запоминаем место, куда загружен контейнер

places[j].occupied = true; // Занимаем место

break; // Переход к следующему контейнеру

}

}

}

}

void ship\_loading::print\_result() {

int total\_profit = 0;

std::cout << "Загруженные контейнеры:\n";

for (int i = 0; i < 8; i++) {

if (loaded[i] != -1) { // Если контейнер загружен

std::cout << "Контейнер " << containers[i].id

<< " -> Место " << loaded[i] + 1

<< " | Вес: " << containers[i].weight

<< " кг | Доход: " << containers[i].profit << " у.е.\n";

total\_profit += containers[i].profit;

}

}

std::cout << "Общий доход: " << total\_profit << " у.е.\n";

}

}

**Main.cpp**

#include <iostream>

#include "Combi.h"

int main() {

setlocale(LC\_ALL, "rus");

combi::ship\_loading loader;

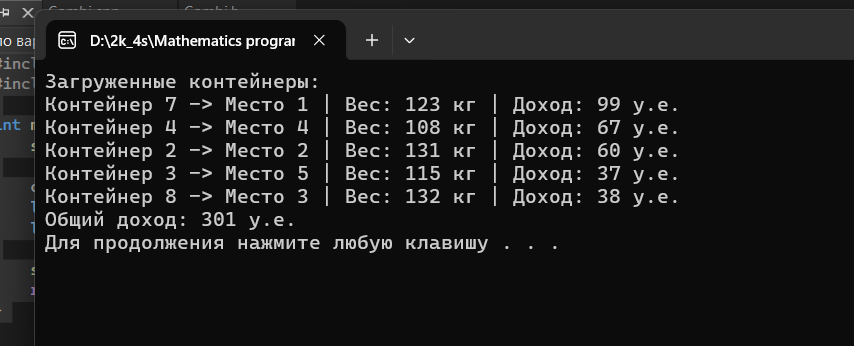
loader.optimize\_loading();

loader.print\_result();

system("pause");

return 0;

}

****

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом, мой 12) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет:

об оптимальной загрузке судна с условием центровки (количество мест на судне для контейнеров 4 – 8);

ex6.h

#pragma once

#include <ctime>

#include <cstdlib>

struct Container {

int weight;

int profit;

};

struct Place {

int min\_weight;

int max\_weight;

};

class ShipLoad {

public:

ShipLoad(int num\_places, int num\_containers); // Конструктор, принимающий количество мест и контейнеров

~ShipLoad(); // Деструктор для освобождения ресурсов

int optimize(); // Функция оптимизации загрузки корабля, возвращающая результат оптимизации

private:

int num\_places;

int num\_containers;

Container\* containers; // Указатель на массив контейнеров

Place\* places; // Указатель на массив мест на корабле

};

Ex6.cpp

#include "6ex.h"

#include <iostream>

ShipLoad::ShipLoad(int num\_places, int num\_containers) {

this->num\_places = num\_places;

this->num\_containers = num\_containers;

srand(time(0));

// Создание контейнеров

containers = new Container[num\_containers];

for (int i = 0; i < num\_containers; i++) {

containers[i].weight = rand() % 101 + 100; // Вес от 100 до 200 кг

containers[i].profit = rand() % 91 + 10; // Доход от 10 до 100 у.е.

}

// Создание мест на судне

places = new Place[num\_places];

for (int i = 0; i < num\_places; i++) {

places[i].min\_weight = rand() % 71 + 50; // Минимальный вес от 50 до 120 кг

places[i].max\_weight = rand() % 701 + 150; // Максимальный вес от 150 до 850 кг

}

}

ShipLoad::~ShipLoad() {

delete[] containers;

delete[] places;

}

// Простая жадная оптимизация загрузки контейнеров

int ShipLoad::optimize() {

int total\_profit = 0;

int used\_places = 0;

for (int i = 0; i < num\_containers && used\_places < num\_places; i++) {

for (int j = 0; j < num\_places; j++) {

if (containers[i].weight >= places[j].min\_weight &&

containers[i].weight <= places[j].max\_weight) {

total\_profit += containers[i].profit;

used\_places++;

break; // Контейнер загружен, идем к следующему

}

}

}

return total\_profit;

}

Main.cpp

#include <iostream>

#include <fstream>

#include <chrono>

#include "6ex.h"

int main() {

std::ofstream file("results.csv");

file << "время место\n";

for (int places = 4; places <= 8; places++) {

auto start\_time = std::chrono::high\_resolution\_clock::now(); // Засекаем время

ShipLoad ship(places, 8); // Количество контейнеров - 8

ship.optimize();

auto end\_time = std::chrono::high\_resolution\_clock::now(); // Останавливаем таймер

double elapsed\_time = std::chrono::duration<double, std::milli>(end\_time - start\_time).count(); // Время в миллисекундах

std::cout << "Places: " << places << ", Time: " << elapsed\_time << " ms\n";

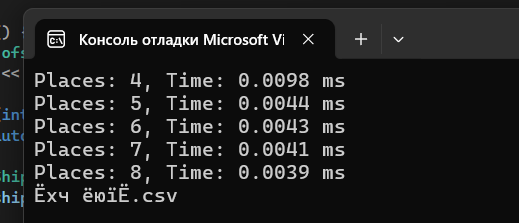
file << places << "," << elapsed\_time << "\n"; // Запись в файл

}

file.close();

return 0;

}



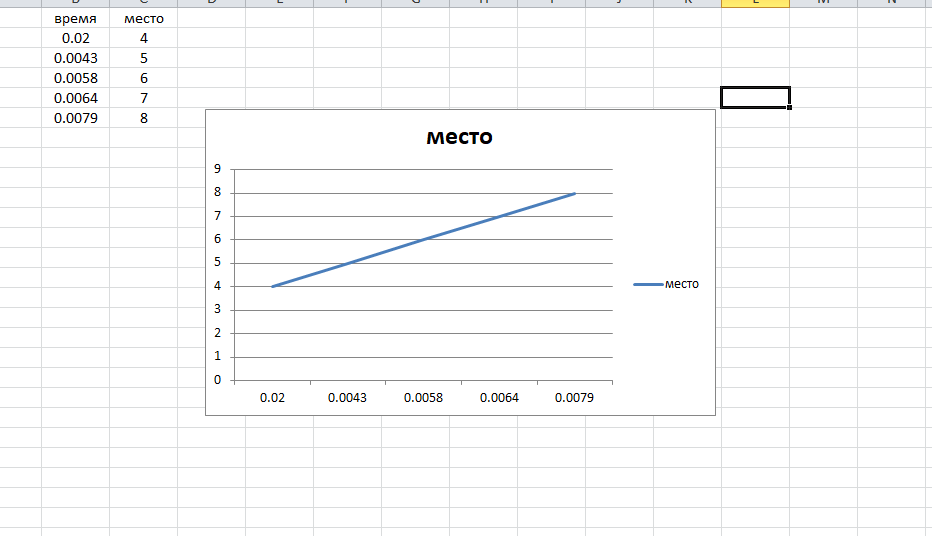


График показывает зависимость времени вычисления задачи оптимальной загрузки судна с центровкой от количества мест на судне.

По мере увеличения количества мест на судне (от 4 до 8) время выполнения алгоритма увеличивается. Линейная зависимость.

1. Что такое комбинаторика?

Комбинаторика — это раздел математики, изучающий способы выбора, упорядочивания и комбинирования объектов из заданного множества. Она включает в себя различные методы подсчета и анализа структур, таких как перестановки, сочетания и разбиения.

2. Что такое генератор?

Генератор в контексте комбинаторики — это алгоритм или программа, которая создает все возможные комбинации, перестановки или другие структуры из заданного множества. Генераторы могут быть использованы для перебора всех вариантов при решении комбинаторных задач.

3. Чем размещения отличаются от сочетаний?

Размещения и сочетания — это два различных способа выбора элементов из множества. Размещения учитывают порядок элементов (то есть разные порядок элементов считается разными размещениями), тогда как сочетания не учитывают порядок (разные порядок элементов считается одним и тем же сочетанием). Например, для множества {A, B, C} размещения из двух элементов будут: AB, AC, BA, BC, CA, CB, а сочетания — только AB, AC, BC.

4. Каких подмножеств множества любой размерности всегда по одному?

Подмножества пустого множества (размерность 0) всегда одно — это само пустое множество. Также существует одно подмножество для множества, состоящего из одного элемента.

5. С помощью какого генератора решается задача о загрузке судна?

Задача о загрузке судна (или задача о **рюкзаке**) часто решается с помощью генераторов сочетаний или динамического программирования. Генераторы позволяют перебрать все возможные комбинации предметов для нахождения оптимального решения.

6. С помощью какого генератора решается задача о **рюкзаке**?

Задача о рюкзаке может быть решена с помощью генераторов сочетаний или динамического программирования. Генераторы могут использоваться для перебора всех возможных комбинаций предметов и их веса, чтобы найти максимальную ценность, которую можно поместить в рюкзак.

7. С помощью какого генератора решается задача о коммивояжере?

Задача о коммивояжере обычно решается с помощью генераторов **перестановок**. Генераторы позволяют перебрать все возможные маршруты (перестановки городов) и выбрать тот, который имеет минимальную длину или стоимость.