ЗВІТ

про виконання лабораторної роботи №3

«Комбінаторика»

з дисципліни «Програмування дискретних структур»

Студента групи КН-2327Б

Сухар Станіслав

Мета роботи: Поглибити і закріпити розуміння основних понять комбінаторного аналізу, формувати навички складання алгоритмів та програм комбінаторного аналізу.

Завдання: Скласти програми із зазначеними вхідними даними та результатами.

1. Обчислити кількість усіх розміщень без повторень з *n* елементів по *r*.

## Варіант:





1.1Текст програми:

Source.cpp

#include <iostream>

using namespace std;

int factorial(int num) {

int result = 1;

for (int i = 1; i <= num; ++i) {

result \*= i;

}

return result;

}

int arrangements(int n, int r) {

return factorial(n) / factorial(n - r);

}

int main() {

setlocale(LC\_CTYPE, "ukr");

int n = 8;

int r = 8;

cout << "Кiлькiсть розмiщень " << n << " елементiв по " << r << ": ";

cout << arrangements(n, r) << endl;

return 0;

}

* 1. Результат виконання коду:

****

* 1. UnitTest

#include "pch.h"

#include "CppUnitTest.h"

#include "../lab3/Source.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace YourTestNamespace

{

TEST\_CLASS(UnitTests)

{

public:

TEST\_METHOD(TestFactorial)

{

Assert::AreEqual(1, factorial(0));

Assert::AreEqual(1, factorial(1));

Assert::AreEqual(2, factorial(2));

Assert::AreEqual(6, factorial(3));

Assert::AreEqual(24, factorial(4));

Assert::AreEqual(120, factorial(5));

}

TEST\_METHOD(TestArrangements)

{

Assert::AreEqual(3024, arrangements(9, 4));

Assert::AreEqual(1, arrangements(5, 0));

Assert::AreEqual(20, arrangements(5, 2));

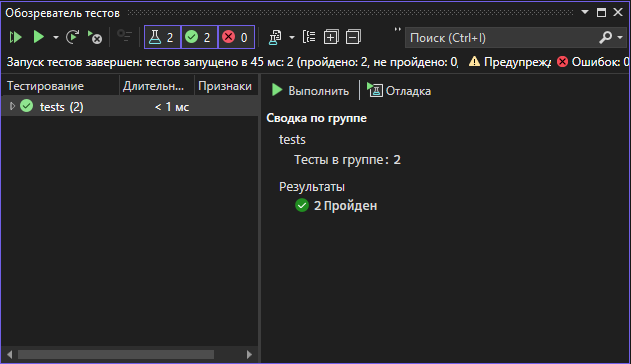
Assert::AreEqual(120, arrangements(5, 5));

}

};

}

* 1. Результат UnitTest:



* 1. Побудова cтруктурної діаграми:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

2.1 Текст програми:

#include <iostream>

#include <iomanip>

using namespace std;

void stirlingNumbers(int n, int F[20][20]) {

for (int i = 1; i <= n; ++i) {

F[i][1] = 1;

F[i][i] = 1;

}

for (int i = 3; i <= n; ++i) {

for (int j = 2; j < i; ++j) {

F[i][j] = F[i - 1][j - 1] + j \* F[i - 1][j];

}

}

}

void bellNumbers(int n, int F[20][20], int B[20]) {

for (int i = 1; i <= n; ++i) {

B[i] = 0;

for (int j = 1; j <= i; ++j) {

B[i] += F[i][j];

}

}

}

int main() {

setlocale(LC\_CTYPE, "ukr");

int variant;

cout << "Ваш номер варiанту: ";

cin >> variant;

int n = variant + 5;

int F[20][20] = { 0 };

stirlingNumbers(n, F);

int B[20] = { 0 };

bellNumbers(n, F, B);

cout << "Числа Стiрлiнга другого роду:\n";

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= i; ++j) {

cout << "| " << setw(3) << F[i][j] << " ";

}

cout << "|\n";

}

cout << "\nЧисла Белла:\n";

for (int i = 1; i <= n; ++i) {

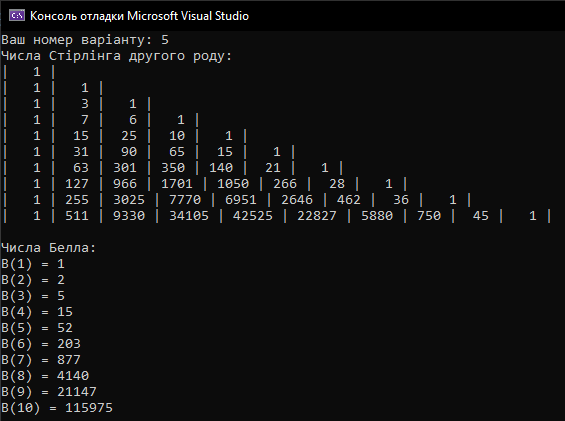
cout << "B(" << i << ") = " << B[i] << endl;

}

return 0;

}

2.2 Результат виконання коду:



* 1. UnitTest:

#include "pch.h"

#include <CppUnitTest.h>

#include "../lab3\_2/Source1.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MyTestProject

{

TEST\_CLASS(MyTestClass)

{

public:

TEST\_METHOD(TestStirlingNumbers)

{

int F[20][20] = { 0 };

stirlingNumbers(1, F);

Assert::AreEqual(1, F[1][1]);

stirlingNumbers(2, F);

Assert::AreEqual(1, F[2][1]);

Assert::AreEqual(1, F[2][2]);

stirlingNumbers(3, F);

Assert::AreEqual(1, F[3][1]);

Assert::AreEqual(3, F[3][2]);

Assert::AreEqual(1, F[3][3]);

}

TEST\_METHOD(TestBellNumbers)

{

int F[20][20] = { 0 };

int B[20] = { 0 };

stirlingNumbers(5, F);

bellNumbers(5, F, B);

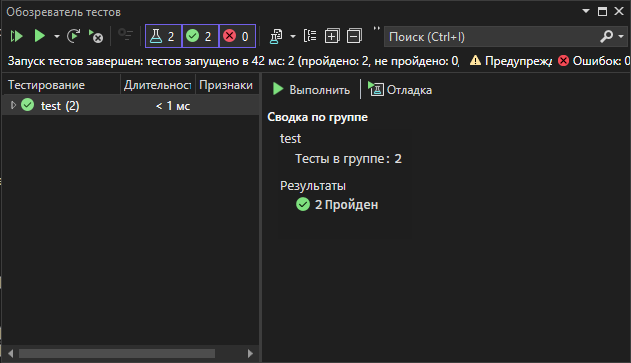
Assert::AreEqual(52, B[5]);

}

};

}

* 1. Результат UnitTest:



* 1. Побудова cтруктурної діаграми:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Посилання на Git-репозиторій:

Висновок:На даній лабораторній роботі, я поглибив і закріпив розуміння основних понять комбінаторного аналізу, сформував навички складання алгоритмів та програм комбінаторного аналізу.