ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ФАКУЛЬТЕТ ЕКОНОМІКИ І МЕНЕДЖМЕНТУ

ФОРМА НАВЧАННЯ ДЕННА

КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА

СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець

(підпис)

«_____»_____2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО БАКАЛАВРСЬКОЇ РОБОТИ

на тему

ПРОГРАМНА РЕАЛІЗАЦІЯ ТРЕНАЖЕРА «ЕЛЕМЕНТИ КОМБІАНТОРИКИ. ФОРМУЛА КЛАСИЧНОЇ ЙМОВІРНОСТІ» ДИСТАНЦІЙНОГО КУРСУ «ТЕОРІЯ ЙМОВІРНОСТІ І МАТЕМАТИЧНА СТАТИСТИКА»

зі спеціальності 122 «Комп'ютерні науки та інформаційні технології»

Виконавець роботи Сіам Карім Ашрафович _____ «___»_____20__ р.

(підпис)

Науковий керівник к. ф.-м. н., доц., Парфьонова Тетяна Олександрівна

_____ «___»_____20___р.

(підпис)

ПОЛТАВА 2020 р.

ЗМІСТ

РЕФЕРАТ

Записка: 33 с., 20 рис., 5 додаток, 8 джерел.

Мета роботи – створення автоматизованої інформаційної **програмна реалізація тренажера «елементи комбіанторики. формула класичної ймовірності» дистанційного курсу «теорія ймовірності і математична статистика».** Сучасний розвиток інформаційних технологій пропонує величезну кількість програмного забезпечення і середовищ для реалізації такого завдання.

Методи, які були використані для розв'язування задачі – програм даних та систем. Робоча програма розроблена в середовищі microsoft Visual Studio 2019.

Розроблено програму «реалізацію тренажеру для вивчення основ мови програмування **C#**»

Здійснена програмна реалізація на мові програмування **C#** в середовищі програмування microsoft Visual Studio 2019.

В результаті тестування, програма показала гарний результат роботи. Можливе подальше вдосконалення програми.

Ключові слова: ТРЕНАЖЕР , МОВА ПРОГРАМУВАННЯ , ***C#***.

ВСТУП

Метою мого дипломного проекту є створення автоматизованої інформаційної системи Елементи тренажера "елементи комбінаторики. Формула класичної ймовірності". Сучасний розвиток інформаційних технологій пропонує величезну кількість програмного забезпечення і середовищ для реалізації такого завдання. Для її вирішення я вибрал середу розробки Microsoft Visual Studio C# 2019 («сі шарп»). До переваг мови C# слід віднести те, що даний мова має 300 000 бібліотек різних функцій, які працюють з максимальною швидкодією. Алгоритми і програми, які представляють інтерес для вивчення і дослідження, обчислюються десятками і сотнями. Одні з них вирішують конкретні завдання на основі відомих методів, інші утворюють "системну" основу для величезної більшості прикладних програм. На сьогоднішній день математичне програмування - важлива складова всього програмування. Великі і складні обчислення завдяки простим програмами стають простими. На сьогоднішній день математичне програмування - важлива складова всього програмування. Великі і складні обчислення завдяки простим програмами стають простими. Microsoft Visual Studio C# 2019. середовища дозволяє розробляти як і звичайні додаток, так і додатки для роботи з систематизованими даними. Дане середовище володіє всіма сучасними можливостями сучасних СУБД (систем управління базами даних).Середовище розробки Visual Studio представляє собою повний набір інструментів для створення як настільних додатків, так і корпоративних веб-додатків. Використовуючи ефективні інструменти розробки Visual Studio, засновані на використанні компонентів, і інші технології, можна не тільки створювати ефективно працюючі настільні додатки, але і спрощувати спільне проектування, розробку і розгортання корпоративних рішень.

1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Поняття Комбінаторики :

Теорія ймовірностей і математична статистика - дві нерозривно пов'язані математичні дисципліни. В даний час їх знання необхідне фахівцям самих різних професій. Уміння формулювати мету своєї діяльності і робити кроки для її досягнення - характерна особливість компетентного, конкурентно здатного фахівця, а теорія ймовірностей і математична статистика, як ніяка інша дисципліна, сприяють позитивним змінам особистості. Знання закономірностей масових випадкових явищ (предмет теорії ймовірностей) і найважливіших методів і прийомів обробки результатів спостережень (вивчає математична статистика) необхідно сучасного програміста при розробці алгоритмів розв'язання практичних завдань. Вивчення ж теорії ймовірностей і математичної статистики немислимо без попереднього знайомства з основами комбінаторики.Комбінаторна математика є старої дисципліною. Вона отримала своє найменування в 1666 р від Лейбніца в його "Dissertation de Arte Combinatori". Комбінаторні алгоритми з їх акцентом на розробку, аналіз та реалізацію практичних алгоритмів є продуктом століття обчислювальних машин.Предмет теорії комбінаторних алгоритмів - обчислення на дискретних математичних структурах. Це новий напрямок досліджень. Лише в останні кілька років з наборів майстерних прийомів і розрізнених алгоритмів сформувалася система знань про розробку, реалізацію та аналізі алгоритмів.Комбінаторика, як можна судити з назви, - розділ математики, що вивчає різні комбінації об'єктів і множин. Комбінаторика дуже тісно пов'язана з інформатикою, і часто зустрічається в олімпіадних задачах.Комбінаторики – ЦЕ розділ математики, пов'язаний з методами підрахунку числа об'єктів певної природи. За змістом завдання зазвичай очевидно, що існує лише кінцеве число цікавлять нас об'єктів. вся справа в тому, щоб знайти це число.Розглянуті об'єкти, як правило, є певними комбінаціями інших об'єктів (чисел,букв і т.д.). Звідси і назва - комбінаторика. У більш широкому розумінні комбінаторика - це теорія кінцевих множин; ми тут розглядатимемо тільки завдання підрахунку, та що таке розширене тлумачення нам не буде потрібно Зі сказаного ясно, що комбінаторика має справу лише з натуральними числами. Може здатися, що вона тому більш «елементарна», ніж інші розділи математики, які оперують з багатим числовим матеріалом (Негативні числа, дробові, ірраціональні, комплексні ...). Але таке судження було б поспішним. Практика показує, що багато, вперше стикаючись з комбінаторики, важко звикають до комбінараторних міркувань (ближчим до програмування, ніж, скажімо, до геометрії). Наша мета полягає в тому, щоб допомогти подолати ці

труднощі. Кращий спосіб освоєння комбінаторики - рішення задач. починати, природно, треба з найпростіших. Саме про простих, типових (і в той же час найважливіших) завданнях і піде мова нижче. Таким образом, комбинаторика – это раздел математики, изучающий вопросы о том, сколько различных комбинаций, подчиненных тем или иным условиям можно составить из множества заданных объектов. Задачей комбинаторики (или комбинаторной задачей) можно считать задачу размещения объектов заданного множества по специальным правилам и нахождение числа способов таких размещений. Комбінаторика грає важливу роль у вирішенні ряду проблем теорії ймовірностей, кібернетики, обчислювальної техніки та інших областях математики. У початковому навчанні математиці роль комбінаторних задач постійно зростає. Це пов'язано з тим, що в них укладе Цікава сама по собі, комбінаторика важлива і для багатьох інших розділів математики. Її зв'язку з алгеброю і теорією ймовірностей будуть коротко переглянути пізніше.

1.2 Основні поняття комбінаторики

Елементами називаються об'єкти, з яких складені з'єднання.Розрізняють такі три види з'єднань: перестановки, розміщення і поєднання. Перестановками з n елементів називають сполуки, що містять всі n елементів і відрізняються між собою лише порядком елементів.Число перестановок з n елементів знаходиться за формулою P_n=n! де n! (Читається "ен-факторіал") - твір натуральних чисел від 1 до n включно, т. Е. $n!=1*2*3*...*n$ наприклад, $p_6 = 6!=1*2*3*4*5*6=120$ Таким чином, отримані комбінації задовольняють різним умовам.Залежно від правил складання можна виділити три типи комбінацій: перестановки, розміщення, поєднання. Попередньо познайомимося з поняттям факторіала.Твір всіх натуральних чисел від 1 до n включно називають n- факторіалом і пишуть n! .

1.2.1 Перестановки

Комбінація з n елементів, які відрізняються один від одного тільки порядком елементів, називаються перестановками. Перестановки позначаються символом Pn, де n- число елементів, що входять в кожну перестановку. (Р - перша буква французького слова permutation- перестановка). Число перестановок можна обчислити за формулою : $P_n = A_n^n = \frac{n!}{(n-n)!} = n!$.

1.2.2 Перестановки з повтореннями

Перестановки з повтореннями - впорядковані множини, в яких елемент a_1 повторюється n_2 раз, a_2 повторюється n2 раз, ..., a_k повторюється n_k раз.

Затвердження. Число всіх перестановок з повтореннями $Pn(n1,n2,\ldots,nr)$ визначається за формулою: $P(n_1, n_2, ...n_k) = \frac{n!}{n_1!, n_2!...n_k}$

де n= n1+ n2+…+n_k.

Доведення. Розглянемо рядок з n позиціями. виберемо
n_i місць, на які поставимо елемент a_1 . Даний-вибір можна здійснити C_n^{m2} способами.
Після цього залишиться n - n1 вільних позіцій, серед яких вибираємо C_{n-n1}^{m2} a_2 способами місця, на які поставимо елемент a_2. Здійснюючи такий вибір k до раз, отримаємо

формулу: $P_n(n_1, n_3, n_3, ..., n_k = C^{n_k}_{n-n1-n_2-...n_{k-1}} = \frac{n!}{n_1! n_2 .... n_k!}$.

1.2.3 Розміщення

В комбінаториці, розміщенням із n елементів по m, або впорядкованою (n, m) вибіркою із множини M (потужність n, m≤n) називають довільний кортеж $(a_{i1}, a_{i2}, ..., a_{im})$ що складається із m попарно відмінних елементів. Розміщення можна розглядати як різнозначну функцію $f;1,2,..., m \to M$ для якої $f(j) = a_{ij}$. Кількість розміщень із n по m позначається як A_n^m або $P(n,m)$ і обчислюється за такою формулою:

$$A_n^m = \frac{n!}{(n-m)!}.$$

Доказательствоь Кожне розміщення являє собою упорядкований набір з m елементів. Перший елемент Можемо вібрато n способамі. второй – n-1 способом ... ", m-й - n - m + 1 способом.Таким чином, за правилом произведения $A_m^n = n.(n-1)...(n-m+1)$**.**

1.2.4 Розміщення з повтореннями

Розміщення з повтореннями - впорядковані m -елементні
підмножини n -елементного множини, які відрізняються як
складом, так і порядком проходження елементів.
Затвердження. Число всіх розміщень з повтореннями A_m^n з n елементів по m визначається за формулою: $\tilde{A}_n^m = n^m$
Доведення. Кожне розміщення являє собою упо-:

рядоченний набір з m елементів. Перший елемент можемо вибрати n способами, другий - n способами, ..., m -й - n способами. Таким чином, за правилом твори

$$\tilde{A}_n^m = n^m .$$

1.2.5 Сполучення :

Сполучення - m -елементні підмножини n -елементного множини, які відрізняються тільки складом елементів (порядок їх слідування не важливий!).

Затвердження. Число всіх сполучень C_m^n. З n елементів по m (Де m < n), визначається за формулою: $C_m^n = \frac{m!}{n!(m-n)!}$.

Доведення. Легко помітити, що $C_m^n = \frac{A_m^n}{P_m}$

Розглянуті вище комбінації відносяться до так званого вибору без повернення - що входять до їх складу елементи не повторюються. Крім того, комбінаторика розглядає і випадки з повторенням елементів, що входять в розглянуті множини - так званий **вибір з поверненням** - розміщення, поєднання та перестановки з n елементів по m , в яких деякі елементи (Або все) можуть бути однаковими.

1.2.6 Сполучення з повтореннями :

Сполучення з повтореннями - m -елементні підмножини n -елементного множини , які відрізняються тільки складом елементів (порядок їх слідування не важливий).

Затвердження. Число всіх сполучень з повтореннями C_m^n з n елементів по m визначається за формулою: $\overline{C}_{n+m-1}^m = \frac{(n+m-1)}{m!(n-1)!}$.

Доведення. «Закодируем» поєднання з повтореннями послідовністю нулів і одиниць наступним чином: спочатку запишемо стільки одиниць, скільки разів в поєднання входить перший елемент вихідної множини, потім - 0, потім стільки одиниць, скільки раз в поєднання входить другий елемент вихідної множини, потім - 0 і т. д. (після одиниць відповідних останнього елемента О не ставимо).

Зауважимо, що в послідовності, що кодує рівно n + m – 1 елемент: т одиниць (стільки, скільки елементів в поєднанні) і n – 1 нулів (відокремлюють один від одного п наборів одиниць, відповідних п елементів вихідного множини). Очевидно, що такий послідовності відповідає деяке сполучення з повтореннями з n по m . Таким чином, встановлено взаємно однозначна відповідність між безліччю поєднань з n по m і безліччю послідовностей з m одиниць і n - 1 нуля. Затвердження доведено.

1.3. Мова програмування C

Вибір мови програмування визначає різноманітність можливостей, які програміст може реалізувати в додатку, а також те, наскільки швидко він це зробить. Останнім часом C і C++ є найбільш використовуваними мовами для розробки комерційних і

бізнес додатків. Ці мови влаштовують багатьох розробників, але в дійсності не забезпечують належної продуктивності розробки. Наприклад, процес написання програми на C ++ часто займає значно більше часу, ніж розробка еквівалентного додатки, скажімо, на Visual Basic. Зараз існують мови, що збільшують продуктивність розробки за рахунок втрати в гнучкості, яка так звична і необхідна програмістам на C / C ++ ,Подібні рішення є дуже незручними для розробників і часто пропонують значно менші можливості. Ці мови також не орієнтовані на взаємодію з з'являються сьогодні системами і дуже часто вони не відповідають існуючій практиці програмування для Web. Багато розробники хотіли б використовувати сучасну мову, який дозволяв би писати, читати і супроводжувати програми з простотою Visual Basic і в той же час давав міць і гнучкість C ++, забезпечував доступ до всіх функціональних можливостей системи, взаємодіяв б з існуючими програмами і легко працював з виникаючими Web стандартами. З огляду на всі подібні побажання, Microsoft розробила нову мову - C #. У нього входить багато корисних особливостей - простота, об'єктна орієнтованість, типова захищеність, "збірка сміття", підтримка сумісності версій і багато іншого. Дані можливості дозволяють швидко і легко розробляти програми, особливо COM + додатки і Web сервіси. При створенні C #, його автори враховували досягнення багатьох інших мов програмування: C ++, C, Java, SmallTalk, Delphi, Visual Basic і т.д. Треба зауважити що у зв'язку з тим, що C # розроблявся з чистого аркуша, у його авторів була можливість (якою вони явно скористалися), залишити в минулому всі незручні і неприємні особливості (існуючі, як правило, для забезпечення сумісності), будь-якого з попередніх йому мов . В результаті вийшов дійсно простий, зручний і сучасну мову, за потужністю не поступається C ++, але істотно підвищує продуктивність розробок.

1. ЕТАПИ РОЗРОБКИ ТРЕНАЖЕРА

2.1 Постановка задачі при пректуванні тренажера

Призначення тренажера - навчання оператора, керуючого деяким технічним об'єктом, для якого потрібно виконувати строго задану послідовність дій. Це рішення задач по комбінаторики з допомогою c#

Назвемо сценарієм навчання необхідну послідовність дій оператора. Керуючі елементи (кнопки, вимикачі) можуть, для простоти, мати тільки два стани: увімкнене.

2.2 Обгрунтування вибору C#

C # - об'єктно-орієнтована мова програмування для платформи .NET. Він розроблений у 2000 році Андерсом Хейлсберг, Скоттом Вілтамутом і Пітером Гольде під егідою Microsoft Research. Основним постулатом C # є вислів: «будь-яка сутність є об'єкт». Мова заснований на суворої компонентної архітектури і реалізує передові механізми забезпечення безпеки коду.
C # був створений спеціально для технології ASP.NET. У той же час на C # повністю написана і сама ASP.NET. C # - це повнофункціональний об'єктно-орієнтована мова, який підтримує всі три «стовпи» об'єктно-орієнтованого програмування: інкапсуляцію, успадкування і поліморфізм. Він має прекрасну підтримку компонентів, надійний і стійкий завдяки використанню «збирання сміття», обробки виключень, безпеки типів. Мова C # розроблявся «з нуля» і увібрав в себе багато корисних властивостей таких мов, як C ++, Java, Visual Basic, а також Pascal, Delphi і ін. При цьому необхідність забезпечення сумісності з попередніми версіями була відсутня, що дозволило мови C # уникнути багатьох негативних сторін своїх попередників. Як і Java, C # розроблявся для Інтернет і приблизно 75% його синтаксичних можливостей аналогічні мови програмування Java, його також називають «очищеної версією Java». 10% подібні мови програмування C ++, а 5% - запозичені з мови програмування Visual Basic. Обсяг нових концептуальних ідей в мові C # близько 10%. Виділення і об'єднання кращих ідей сучасних мов програмування робить мову C # не просто сумою їх достоїнств, а мовою програмування нового покоління, тому я вибрав саме цю мову для написання тренажера.

3. ІНФОРМАЦІЙНА ЧАСТИНА

3.1 Зміст тренажера

1. Історичний огляд:

Основні поняття комбінаторики, Розвиток комбінаторики, Поняття ймовірності та зародження науки про закономірності випадкових явищ, Рішення історичних завдань, Множина, логіка, Об'єднання, перетин, доповнення. Висловлювання. Теореми. Математична логіка. Терміни та символи.

3. Введення в комбінаторики

Комбінаторні задачі. Правило суми. Правило множення. Рішення задач за допомогою. Факторіал. Трикутник Паскаля. Біном Ньютона. Перестановки без повторень. Перестановки з повтореннями. Розміщення без повторень. Розміщення з повтореннями. Сполучення без повторень. Сполучення з повтореннями.

4. Випадкові події та їх ймовірності

Події достовірні, неможливі, випадкові. Класичне поняття ймовірних подій. Статистичне поняття ймовірності події. Геометричне поняття ймовірності. Формула Бернуллі.

3.2 Розробка функцій тренажера

Функції для C #, створювані в Visual Studio, дозволяють розширити мову запитів за рахунок власних функцій. Ви можете повторно використовувати вже існуючий код, а також математичну або комплексну логіку з C #. Існує три способи реалізації функцій: Файли коду програмної частини в проекті, призначені для користувача функції з локального проекту C # і призначені для користувача функції з існуючого пакету в обліковій записи зберігання. У цьому керівництві для реалізації базової функції C # використовується метод коду програмної частини. Функції для для завдань зараз знаходяться на етапі попередньої версії, і їх не слід використовувати в робочих навантаженнях.

У цьому керівництві описано наступне:

- Створення користувальницької функції C # за допомогою коду програмної частини
- Тестування завдання на локальному комп'ютері.
- Публікація завдання.

Написання для користувача функції C # за допомогою коду програмної частини

Файл коду програмної частини - це файл C #, пов'язаний з одним сценарієм запитів. Засоби Visual Studio автоматично архівують файл коду програмної частини і після передачі відправляють його до вашого профілю зберігання. Всі класи повинні бути визначені як public, а об'єкти - як static public.

3.3 Описание алгоритм решения задач

При решении комбинаторных задач исследуется сущность комбинаций, а для их исчисления используются правила сложения и умножения, а также формулы, выведенные из этих правил. Начнем с задачи на использование правила сложения,У наведеному протоколі виділено три етапи:. Позначення даних і шуканих (первинні і цільове безлічі, їх чисельності); . Вираз зв'язку даних і шуканих, розбиття цільового множини на класи і їх обчислення (в правому полі), остаточна формула (правило додавання); . Підрахунок шуканого числа за формулою. Етапи і - побудова математичної моделі явища, становлять логічне ядро в рішенні будь-якої математичної задачі. У модель можуть включатися й інші допоміжні безлічі. Відомі числові множини допустимо використовувати без попереднього вказівки на етапі.Спочатку виявляється зв'язок між множинами, виражається у формі опису

елементів цільового множини через елементи первинного і допоміжних множин. Зокрема встановлення зв'язку з цим допомагає виявити вид з'єднань (перестановки, розміщення і т. Д.). Потім проводиться перехід до зв'язку між численностями цих множин за відповідною формулою комбінаторики.У найпростіших завданнях шукається чисельність множини з'єднань - комбінацій заданого числа елементів. Це число r назвемо розміром вибірки. Для перестановок воно дорівнює чисельності первинного. У перелік властивостей входять розмір з'єднання, повторюваність елементів і їх впорядкованість.На етапі в процесі опису похідного множини яких навчають з'ясовують, що його елементи є вибірками з заданим переліком властивостей. По набору розпізнаних властивостей встановлюють, що в даному випадку з'єднання є розміщеннями, записується відповідна формула зв'язку численностей цих множин. Далі на етапі здійснюють підрахунок числа з'єднань цій формулі. Зв'язок в розглянутої моделі можна висловити і по-іншому - задати похідне безліч алгоритмом породження довільного елемента.Метод побудови опису зв'язку між заданими множинами і цільовим безліччю списком властивостей простіше алгоритмічного методу, однак, він не завжди приводить до успіху. Може виявитися, що з'єднання не належить до жодного стандартного вигляду. Вихід зі скрути в переході від списку властивостей до алгоритму.

3.4.Модель створеного математичного тренажера.

Модель математичного тренажера включає в себе виведені на екран монітора визначення комбінаторики, факторіала, правил твори і підрахунку числа перестановок, а також декількох завдань, які пропонуються для самостійного рішення.

Модель тренажера приведена в додатку.

Робота даної моделі математичного тренажера була випробувана на 5 учнів. З них:

- вирішили все 4 запропоновані завдання - 0 осіб
- вирішили 3 з 4 запропонованих завдань - 3 людини
- вирішили 2 з 4 запропонованих завдань - 2 людини

Апробація моделі тренажера навіть на такій невеликій кількості учнів вже дозволяє зробити висновок про те, що його використання при вивченні комбінаторики може бути корисним і суттєво підвищить рівень знань по деяких розділах математики.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Тренажера вирішує наступні завдання

тренажер включає в себе теми для користувачів і студентів, які хочуть вивчити комбінаторики, які воно містить, тренажер допомагає користувачеві зрозуміти такі теми:

- Поняття Комбінаторики
- Основні поняття комбінаторики
- Основна формула комбінаторики
- Елементи комбінаторики
- Перестановки
- Розміщення
- Сполучення

4.2. Опис тренажера

тренажер простий і зручний у використанні. тренажер ділиться на кілька частин, перша частина Містить короткий опис комбінаторики , І їх типи, щоб користувач міг зрозуміти тему, яка обговорювалася в тренажері, і що вона містить.

Друга частина містить п практичну частину, з допомогою якої користувач може вводити значення і отримувати результати. Третя частина містить простий тест, в якому користувач може перевірити себе, і якщо він не знає, як його вирішити, він може натиснути на значок допомогу .

4.3. перевірка чистота коду

Навіщо нам потрібен гарний код?

Зазвичай, коли ми працюємо над конкретним програмним продуктом, естетичні якості коду турбують нас далеко не в першу чергу.
Нам набагато важливіше наша продуктивність, якість реалізації функціоналу, стабільність його роботи, можливість модифікації і розширення і т.д.
Але чи є естетичні якості коду фактором, що позитивно впливає на перераховані вище показники?
Моя відповідь: так, і при цьому, одним з найважливіших!

Це так, тому що гарний код, незалежно від суб'єктивного трактування поняття про красу, володіє наступними (в тій чи іншій мірі зводяться один до одного) найважливішими якостями:

Читаність. Тобто можливість, дивлячись на код, швидко зрозуміти реалізований алгоритм, і оцінити, як буде вести себе програма в тому чи іншому окремому випадку.

Керованість. Тобто можливість в мінімальні терміни внести в код необхідні поправки, уникнувши при цьому різних неприємних передбачуваних і непередбачуваних наслідків.

Чому ці якості дійсно є найважливішими, і як вони сприяють підвищенню показників, зазначених на початку параграфа, впевнений, очевидно кожному, хто займається програмуванням.

А тепер, щоб від загальних слів перейти до конкретики, давайте зробимо зворотний хід і скажемо, що саме читають і керований код зазвичай сприймається нами як гарний і професійно написаний. Відповідно, на обговоренні того, як домогтися цих якостей, ми далі і зосередимося.

Три базові принципи.

Переходячи до викладу власного досвіду, зазначу, що, працюючи над читаністю і керованістю свого і чужого коду, я поступово прийшов до наступного розуміння.

Незалежно від конкретної мови програмування і вирішуваних завдань, для того, щоб фрагмент коду в достатній мірі володів цими двома якостями необхідно, щоб він був:

- максимально лінійним;
- коротким;
- самодокументірованним.

Можна нескінченно перераховувати різні хинти і техніки, за допомогою яких можна зробити код красивіше. Але я стверджую, що найкращих, або, у всякому разі, досить хороших результатів можна досягти, орієнтуючись саме на ці три принципи. Тому далі я постараюся докладно пояснити їх суть, а також описати набір основних технік, за допомогою яких можна привести свій код у відповідність з цими принципами.

4.4. Тестування тренажера

Інструкція користувача : натиснути на праграммо з назві start як указное на рис.1

рис.1

- Під час запуску програми з'являється вікно як на рис.2.

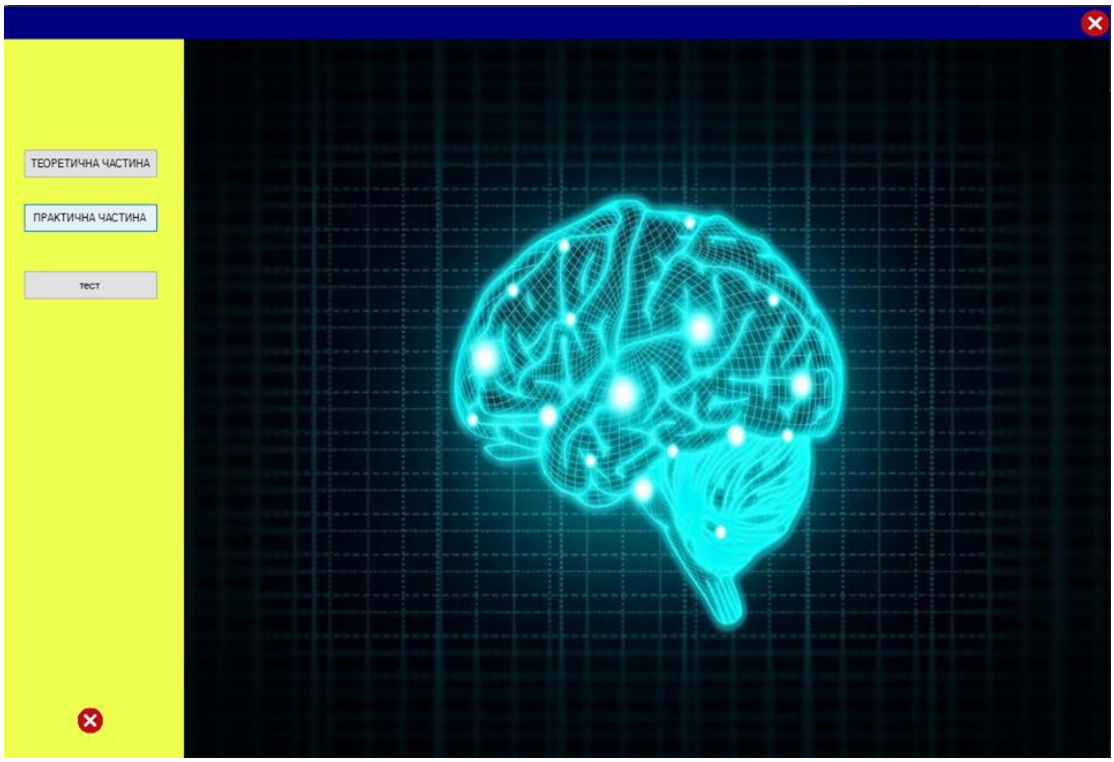


рис. 2 Вікно початку програми

- У вікні містяться три кнопки:
  - " теоретична частина "
  - " практична частина "
  - " тест ".

При натисканні першої кнопки " теоретична частина " відбувається перехід де міститься інформація про теоретична частина про комбінаторики . При натисканні другий відкриється вікно в неї нова форма для практична частина, де міститься практични *завдання про* комбінаторики . Третя кнопка тест.

- Після натискання кнопки теоретична частина на екрані з'являється вікно зображене на рис.3.

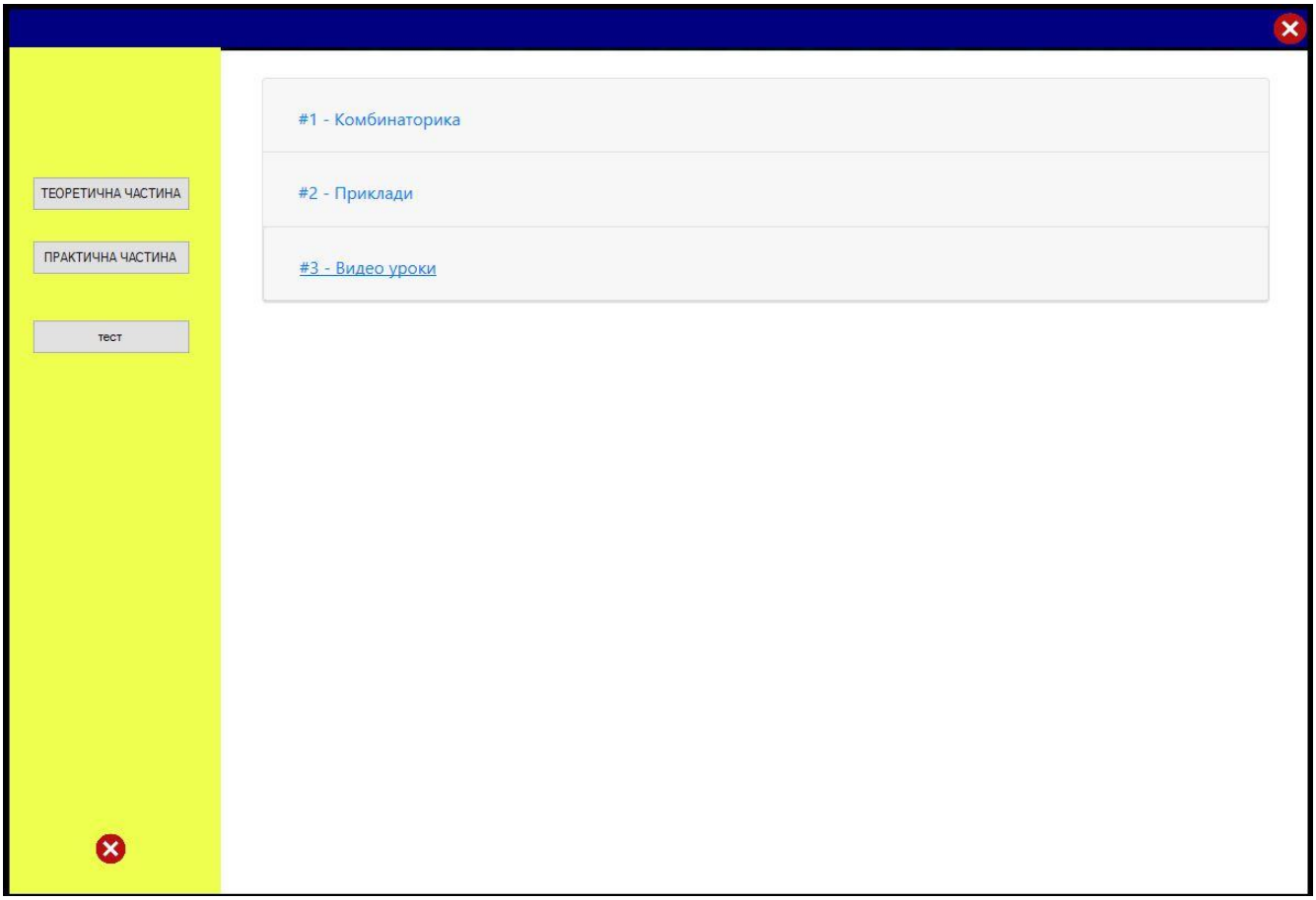


рис. 3

На третьому зображенні ви знайдете теоретична частина, в наступному вікні три кнопки :Перша кнопка показує нам визначення комбінаторики рис. 4 , Друга кнопка призводить нас до прикладів комбінаторики рис. 5 . Третя кнопка - посилання на поіснінія відео на YouTube рис. 6.

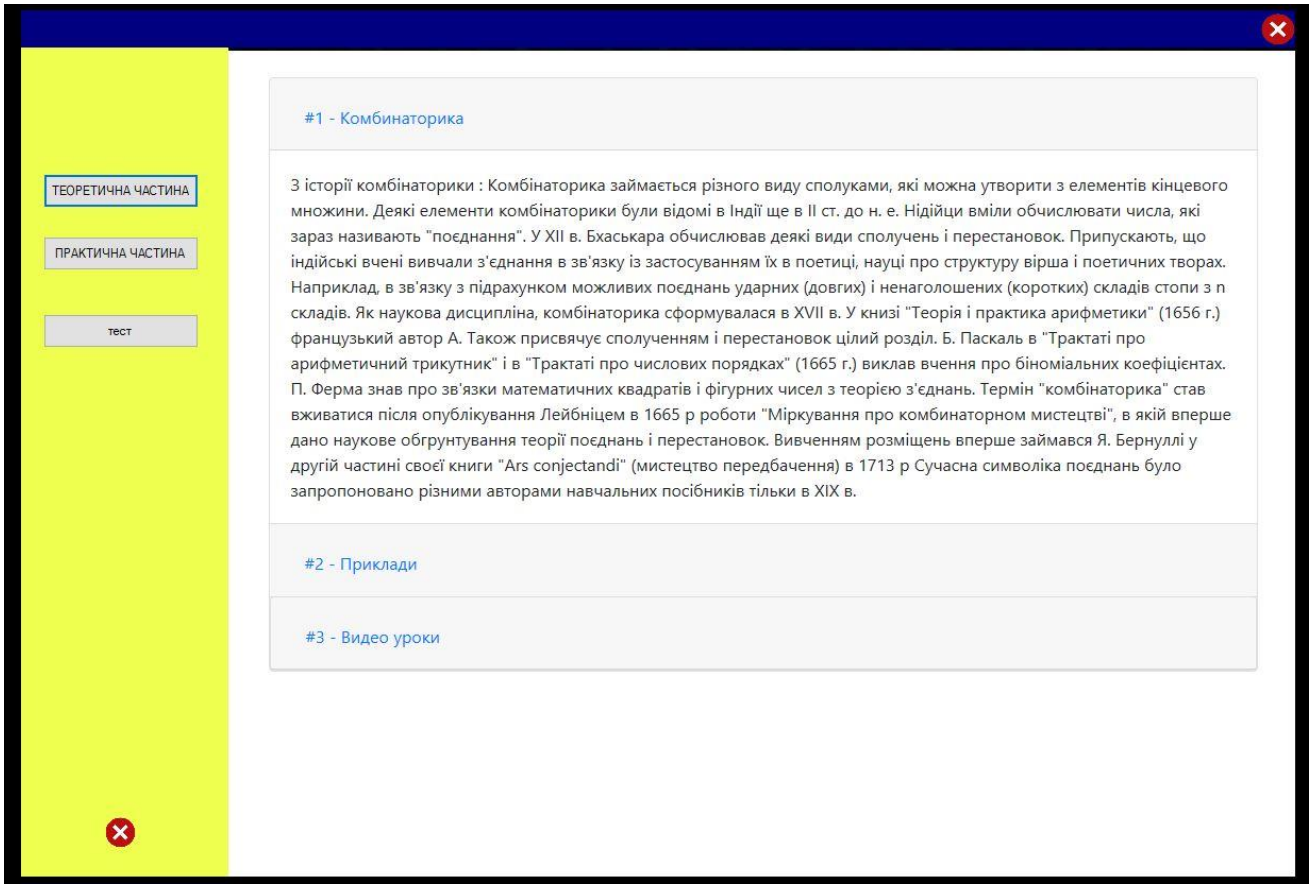


рис. 4 визначення комбінаторики

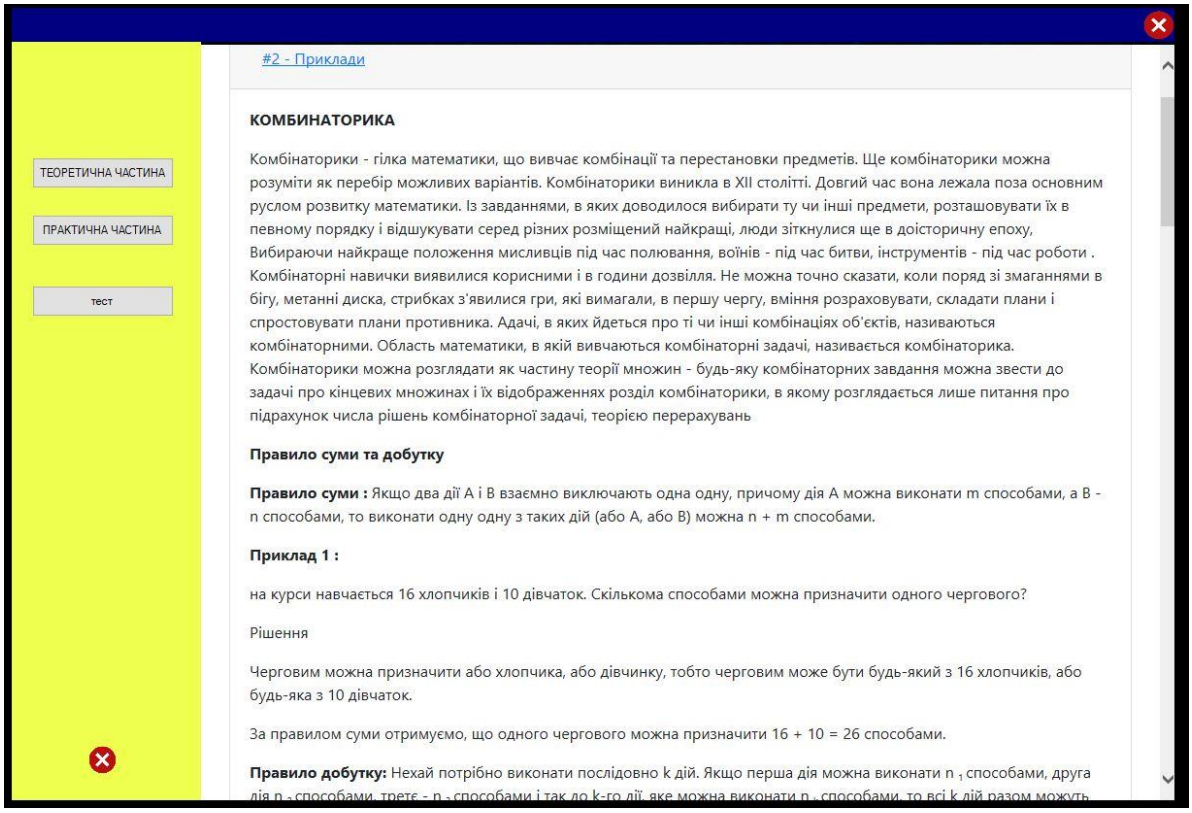


рис. 5 прикладів комбінаторики

Коли ви натискаєте третю кнопку, кнопка «Уроки» на YouTube показує нам наступний зображення рис. 6.

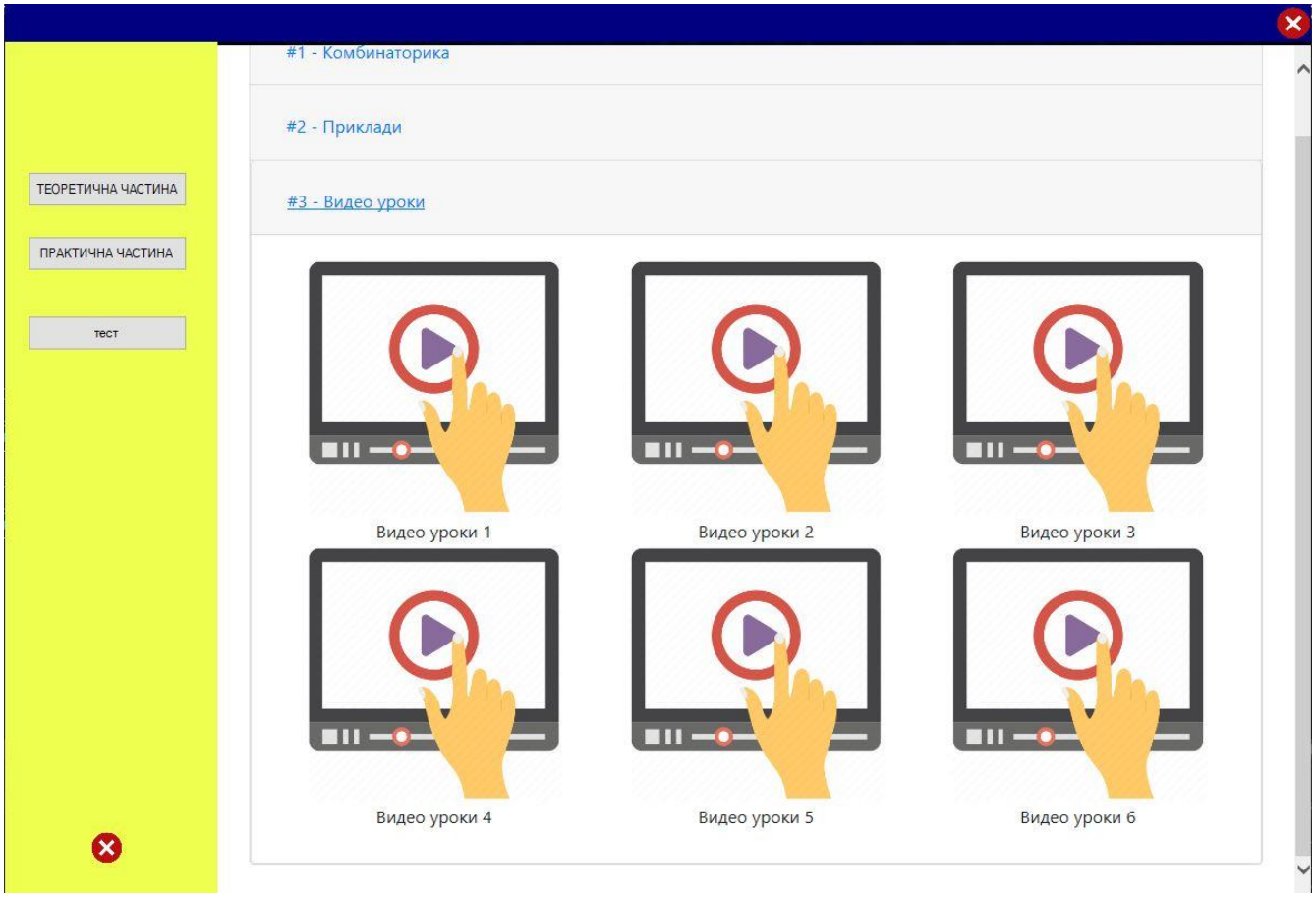


рис. 6 відео уроки на YouTube

Побачивши таке зображення рис**.** 6, ми можемо натиснути на будь-яке відео і вибрати його, для перегляду, і після цього він відкриє нам картинку, як показано на рис**.** 7 , і звідси ми бачимо сторінку YouTube, на якій містяться уроки, пояснюють тему про комбінаторики .

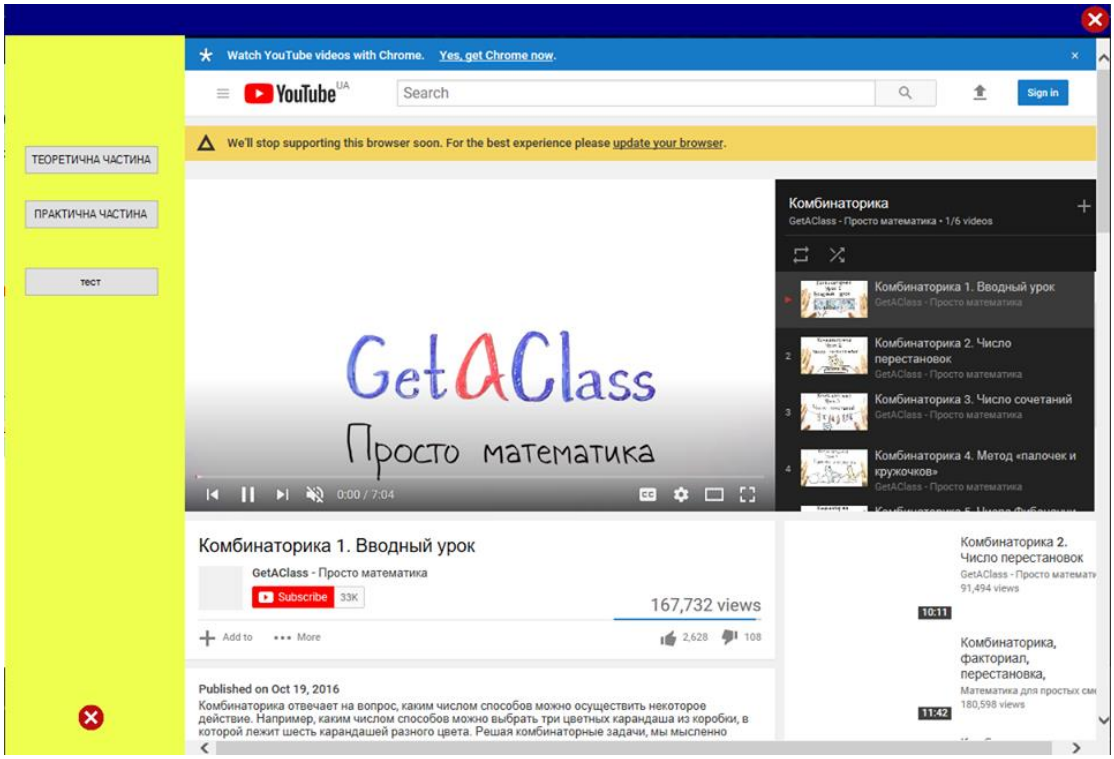


рис. 7 .поіснніння відео на YouTube

При натисканні на теоретична частина , щоб запустити програму, ви повинні натиснути ліву чи праву кнопку на лівій стороні програми рис. 8

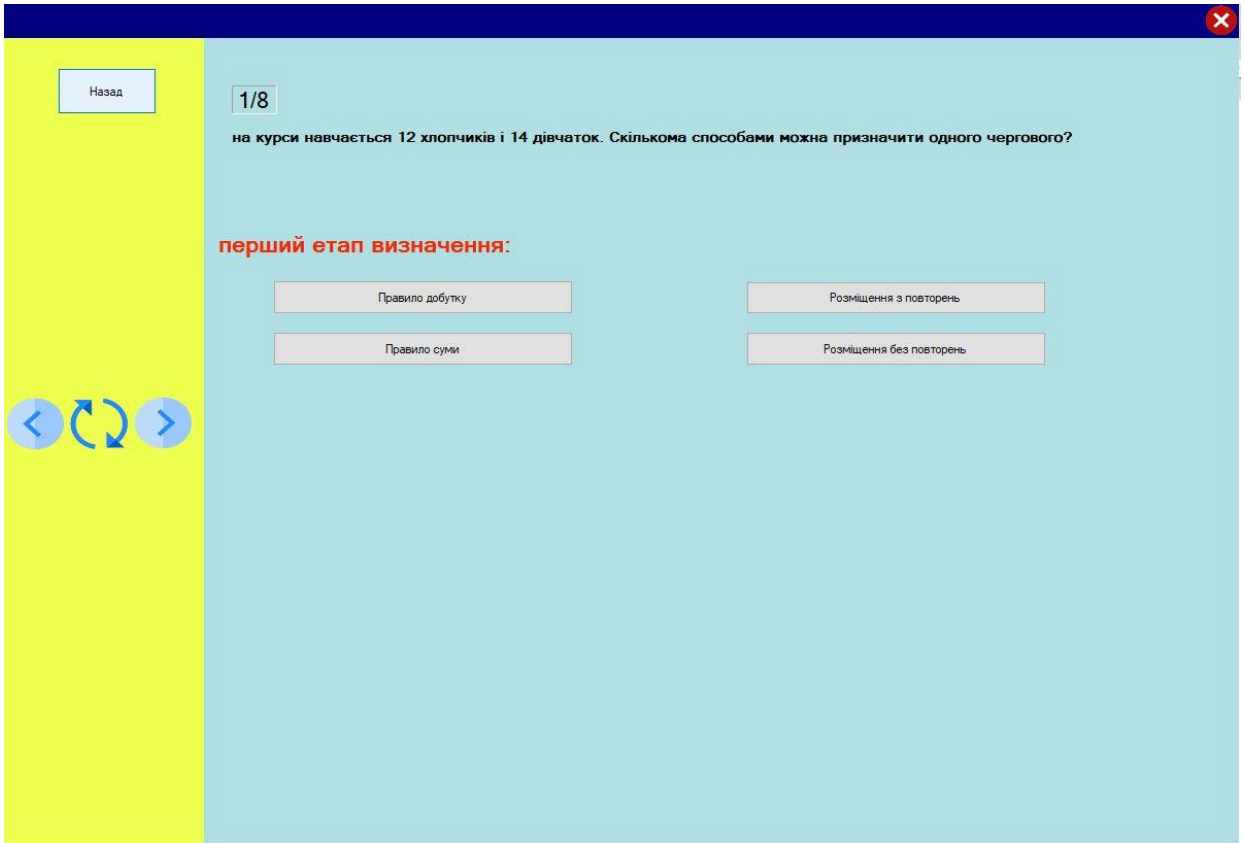


рис. 8 запуск теоретична частинаВін складається з трьох абзаців, перший визначає тип Відкрийте вікно або пояснення статті, щоб пояснювати рис. 9

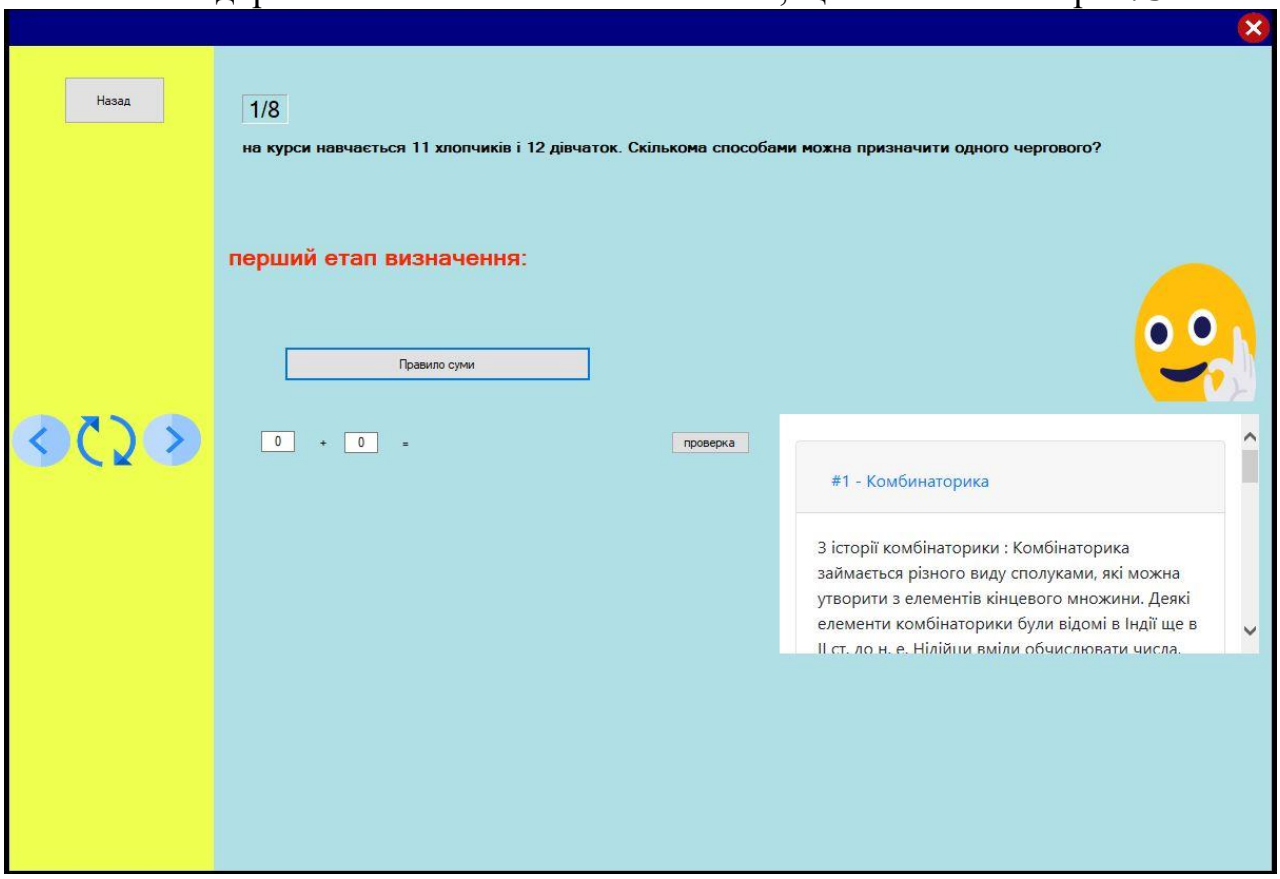


рис. 9

Користувач може вводити інформацію з клавіатури і вирішувати завдання щоб перевірити себе , Якщо відповідь правильна, він покаже нам картинку, яка показана як правильну відповідь.А при відсутності неправильної відповіді картина виглядає як неправильну відповідь рис. 10 рис. 11

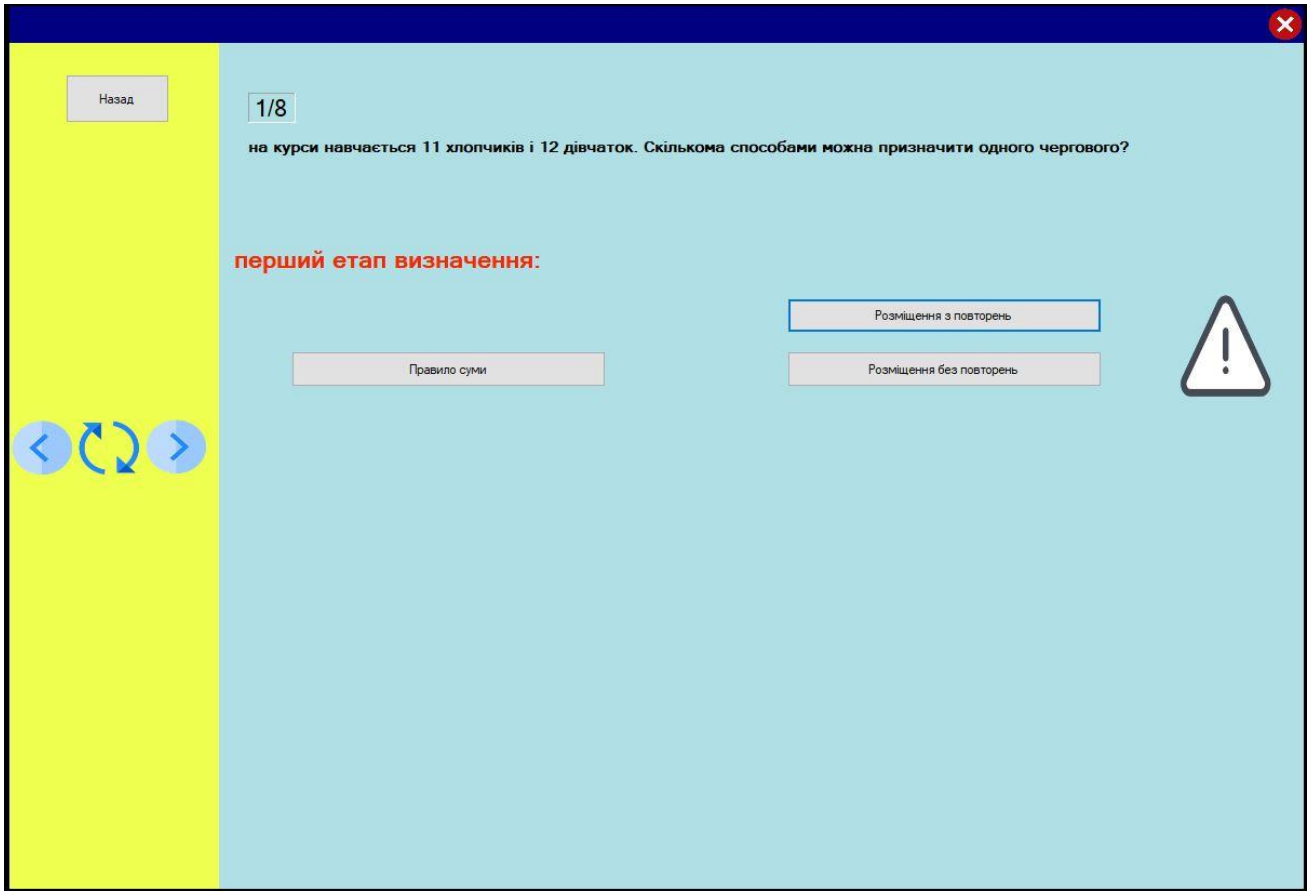


рис. 10 неправильної відповіді

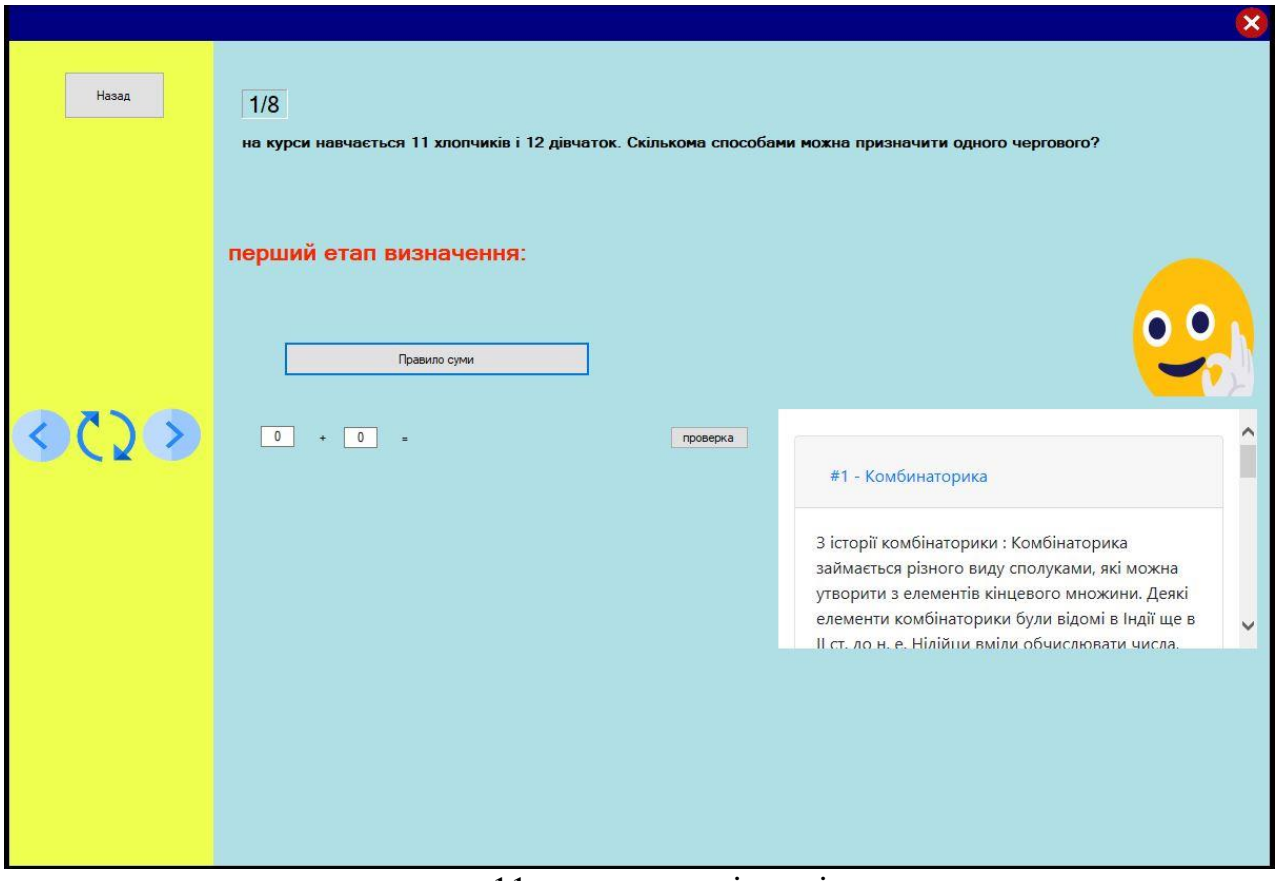


рис. 11 правильну відповідь

Отже, 8 завдань різних на тему комбінаторики, Коли обраний правильну відповідь, з'являється вікно, і це вікно містить теоретична частина і приклад як на рис. 10 Вибравши правильну відповідь і прочитавши теоретична частина і практична частина користувача, може вести інформацію, щоб відповісти на питання як на рис. 12 . У разі, якщо користувач вводить правильну відповідь, з'являється зображення, що представляє його як правильний ответ рис. 12.Что стосується неправильного відповіді, зображення показує, що користувач допустив помилку рис. 13.

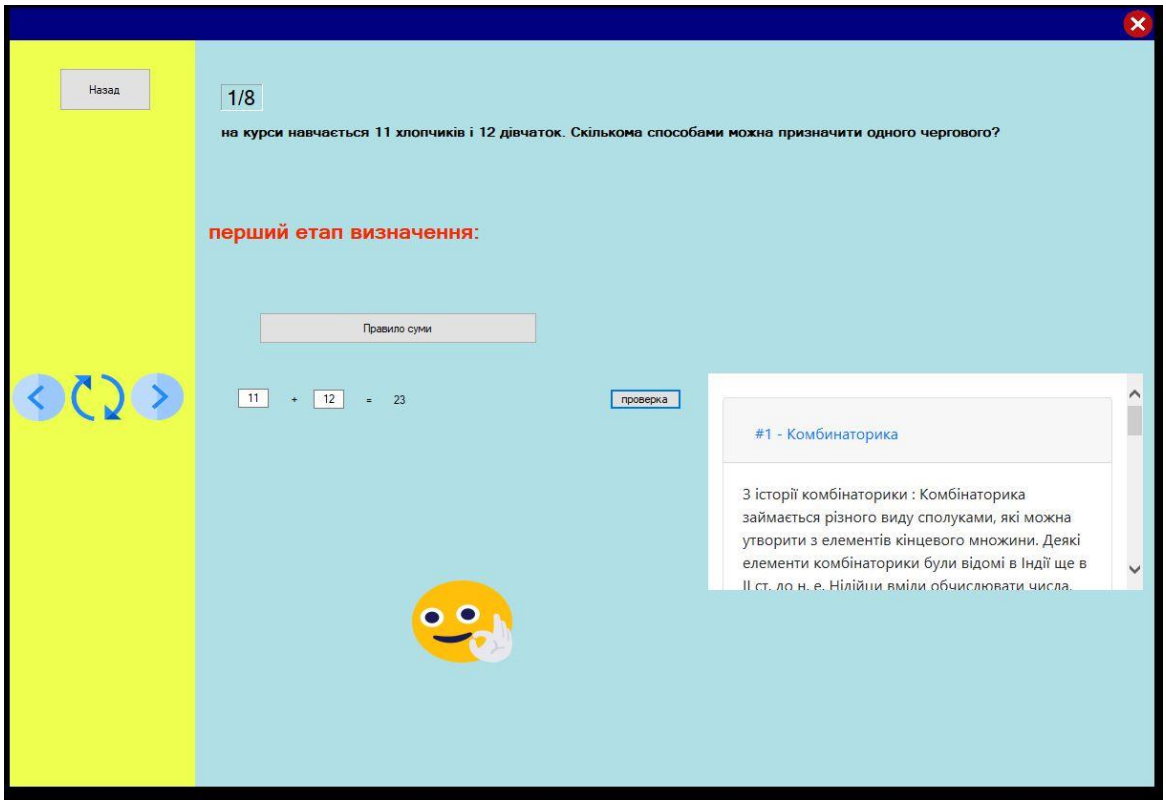


рис. 12 правильну відповідь

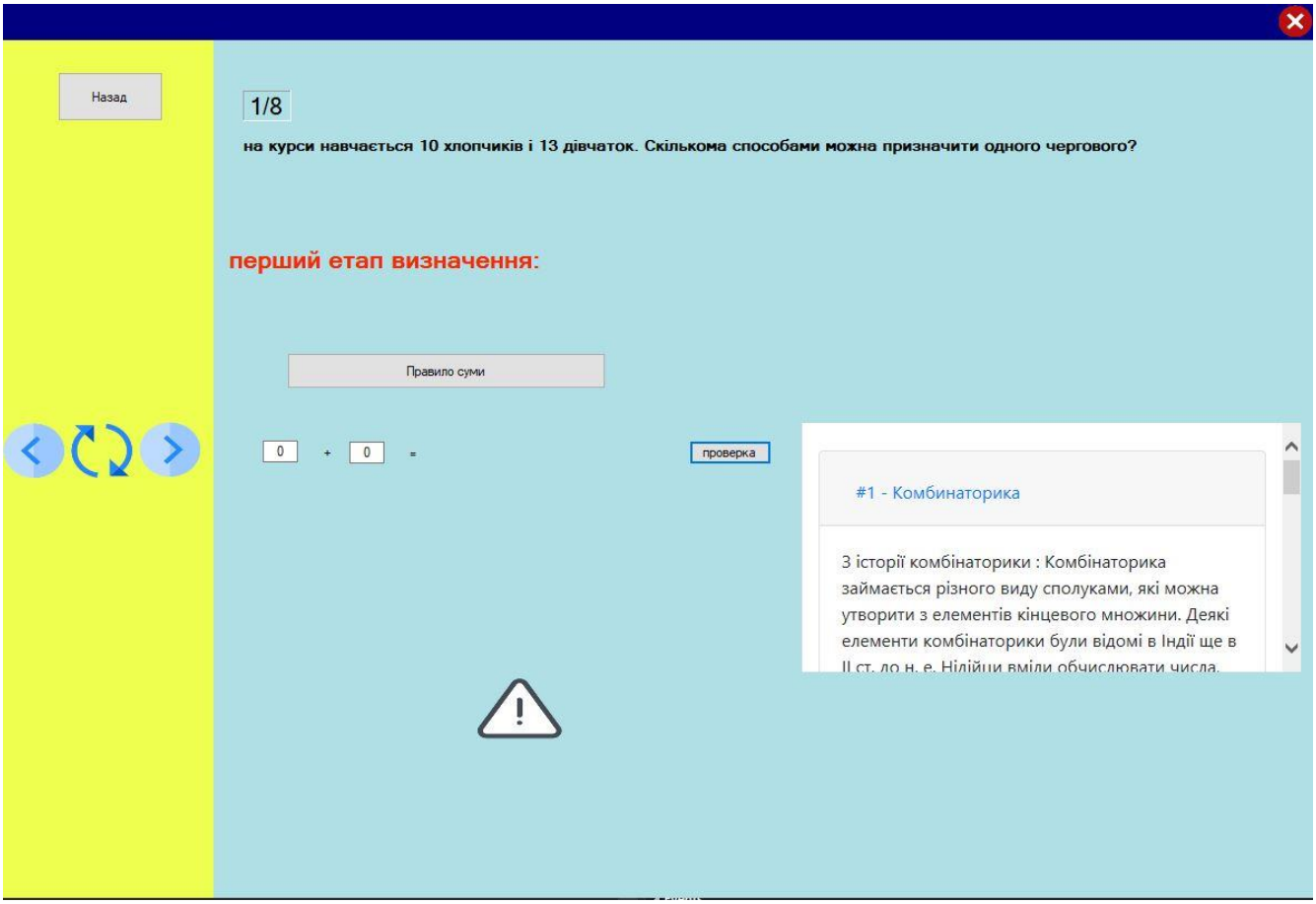


рис. 13 неправильної відповіді

- Щоб почати іспит, ми повинні натиснути кнопку іспиту рис . 14

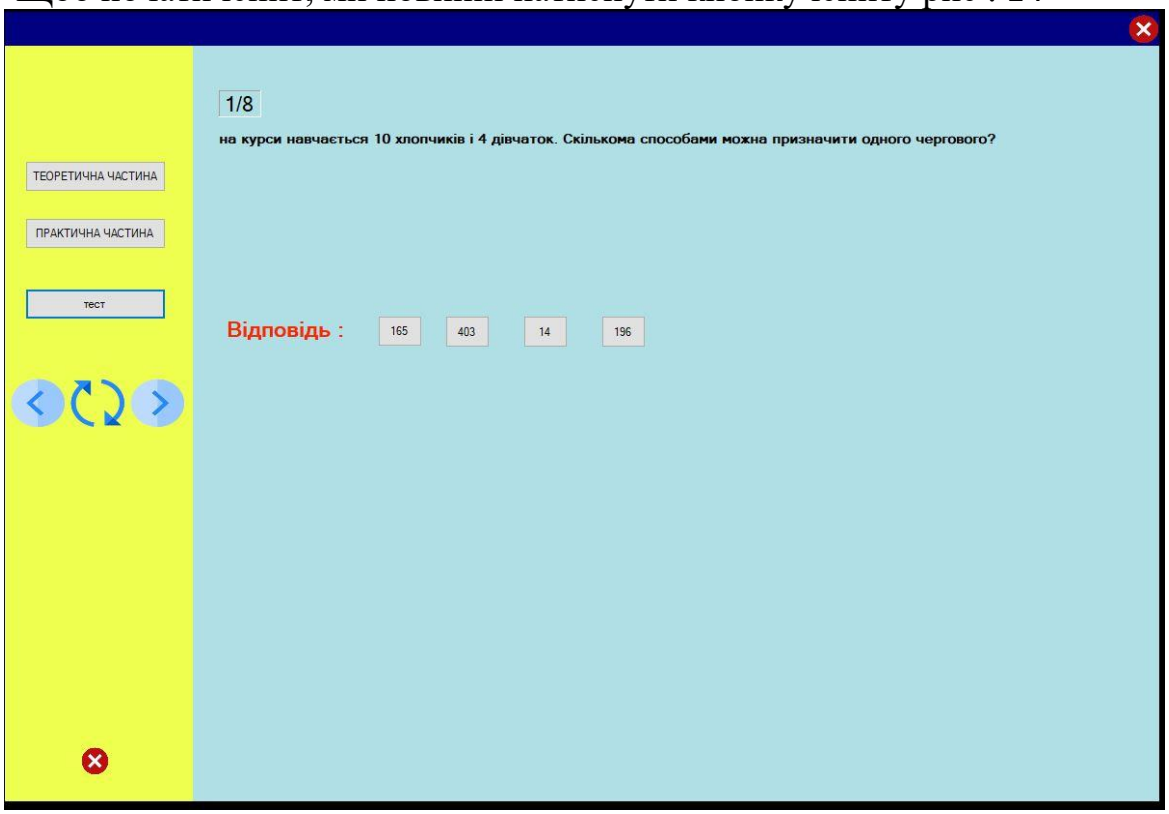


рис. 14

Коли користувач вибирає два різних варіанти, ошібачніх з'являється вікно кнопки допомоги. Користувач може звернутися до теоретична частина і переглянути інформацію рис. 16

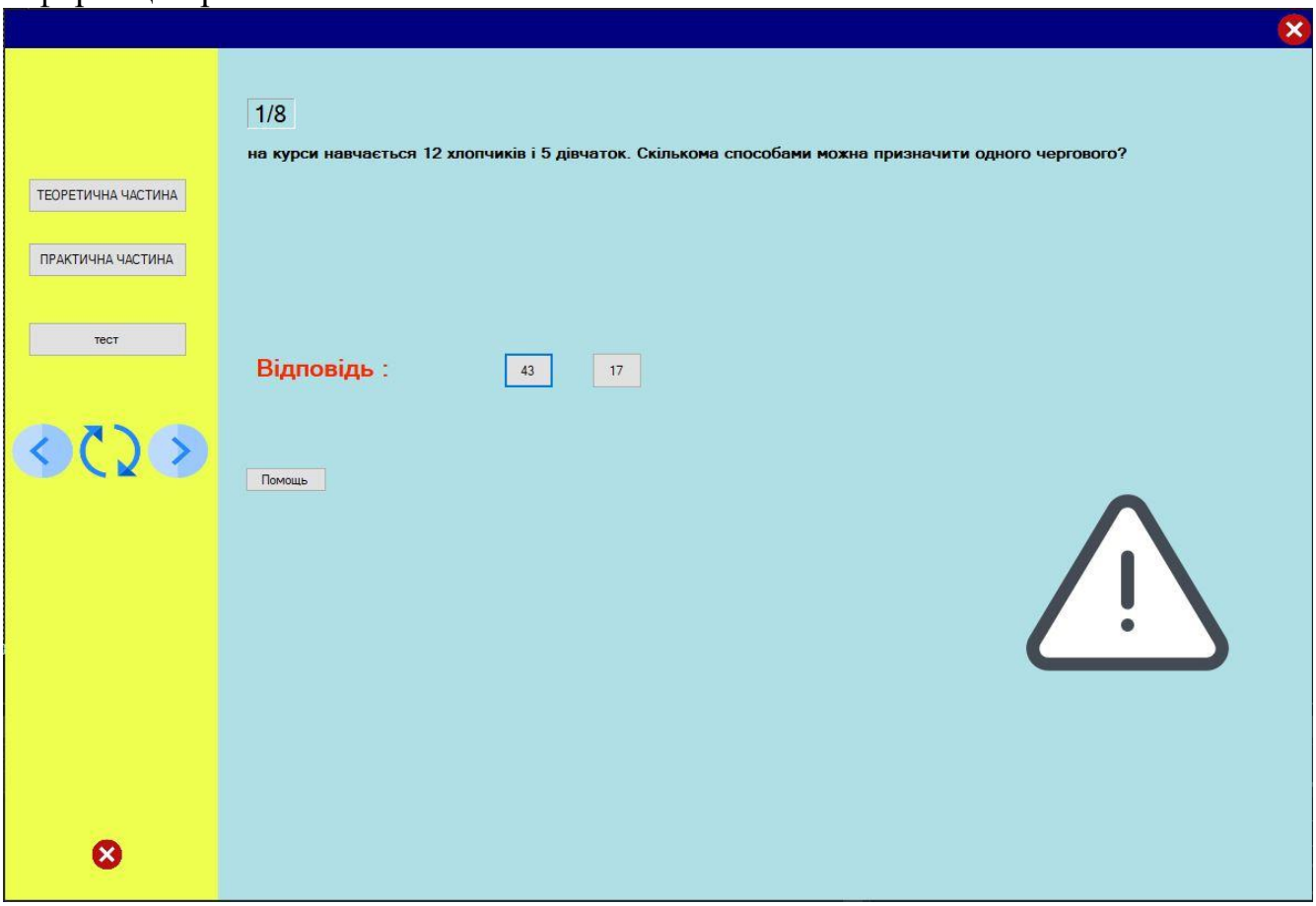


рис. 16

ВИСНОВК

висновок: Комбінаторика і програмування нерівномірно пов'язані в предметі інформатики. Це повинно мати важливе значення в олімпіадному програмуванні. Саме важливим є практичне застосування комбінаторики в програмуванні для вирішення технічних завдань при автоматизації розрахунків кількості можливих ситуацій вивчивши всього лише невелику частину наявної літератури з комбінаторики, я прийшов до висновку про те, що вдосконалення навичок вирішення завдань з даного розділу математики істотно допоможуть мені і моїм колегам-учням освоювати теорію ймовірностей і математичну статистику, так як саме наука про підрахунок числа комбінацій лежить в основі методів вирішення найпростіших імовірнісних задач.Надалі мені б хотілося продовжити роботу над цією тематикою, розробити повноцінний тренажер, а не тільки його модель, куди включити завдання і на підрахунок числа сполучень і розміщень. Якщо ця робота

вдасться, то розроблений тренажер можна буде використовувати, присвячених теорії ймовірностей.

ДОДАТОК Ж
СПИСОК ЛІТЕРАТУРИ

Список використаних інформаційних джерел:

1. Комбінаторна теорія (Айгнер М., 1982).

2. Конкретна математика (Грехем Р., Кнут Д., Паташнік О., пров. З англ., 1998).

3. Рибников К.А. Введення в комбінаторний аналіз.

4. Виленкин Н.Я., Виленкин А.Н., Виленкин П.А. Комбінаторика. - М., 2006. - 400 с.

5. C Sharp. Матеріал з Вікіпедії — вільної енциклопедії. [Електронний ресурс] – Режим доступу: http://uk.wikipedia.org/wiki/C_Sharp – Назва з титул. екрану.

6. [Електронний ресурс] – Режим доступу: Введение в c# https://docs.microsoft.com/en-us/dotnet/csharp/ .

7. Mono. Mono is a cross platform, open source .NET development framework. [Електронний ресурс] – Режим доступу: http://www.mono-project.com/ .

8. Microsoft. Центр загрузки програмное обеспичение. Microsoft Visual Studio [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com/ru/downloads/ .

ДОДАТОК А. Програма

Ієрархія проекту та її Компоненти

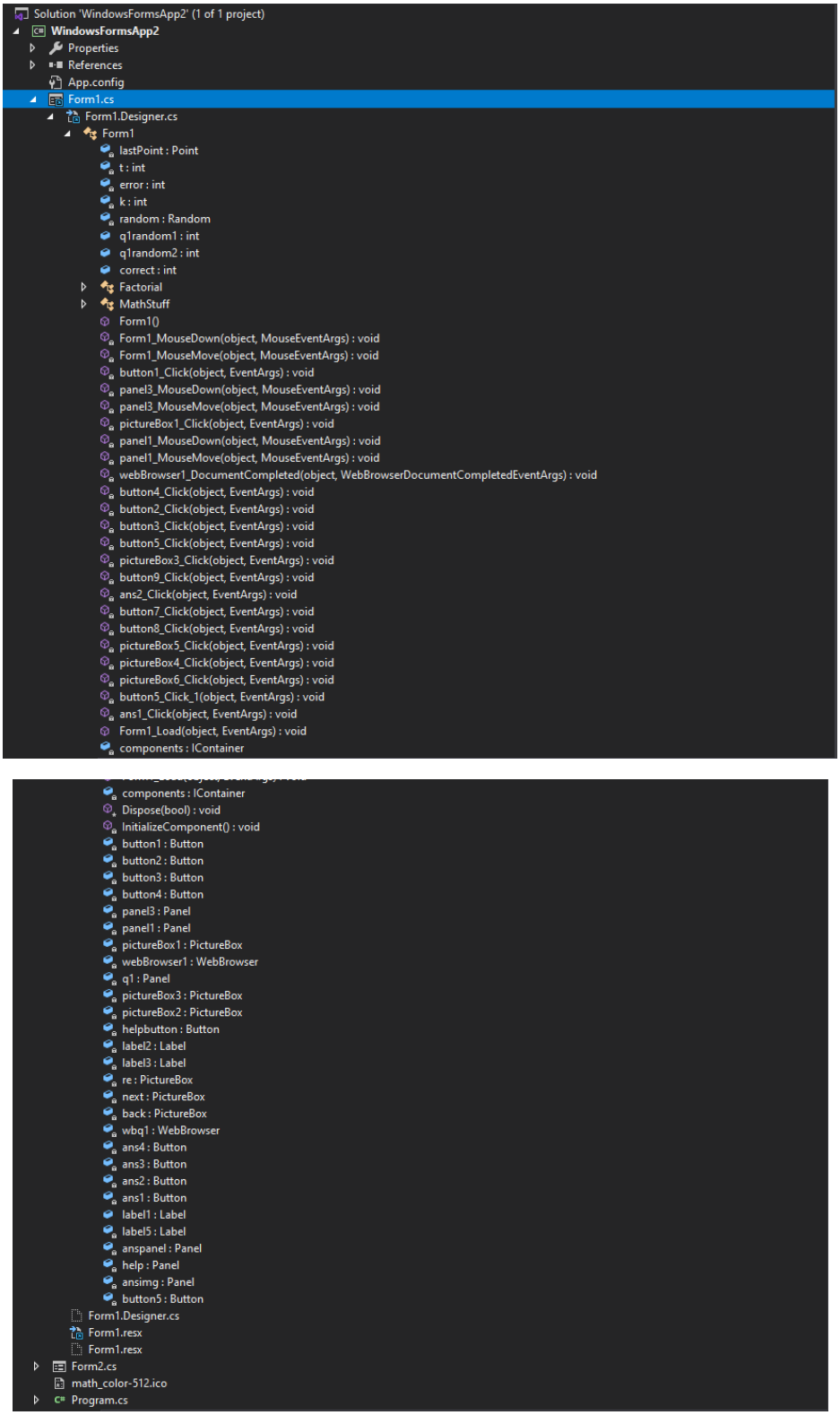


рис.17 ієрархія проекту

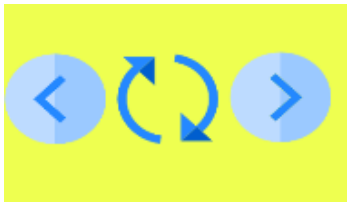


Рис. 18

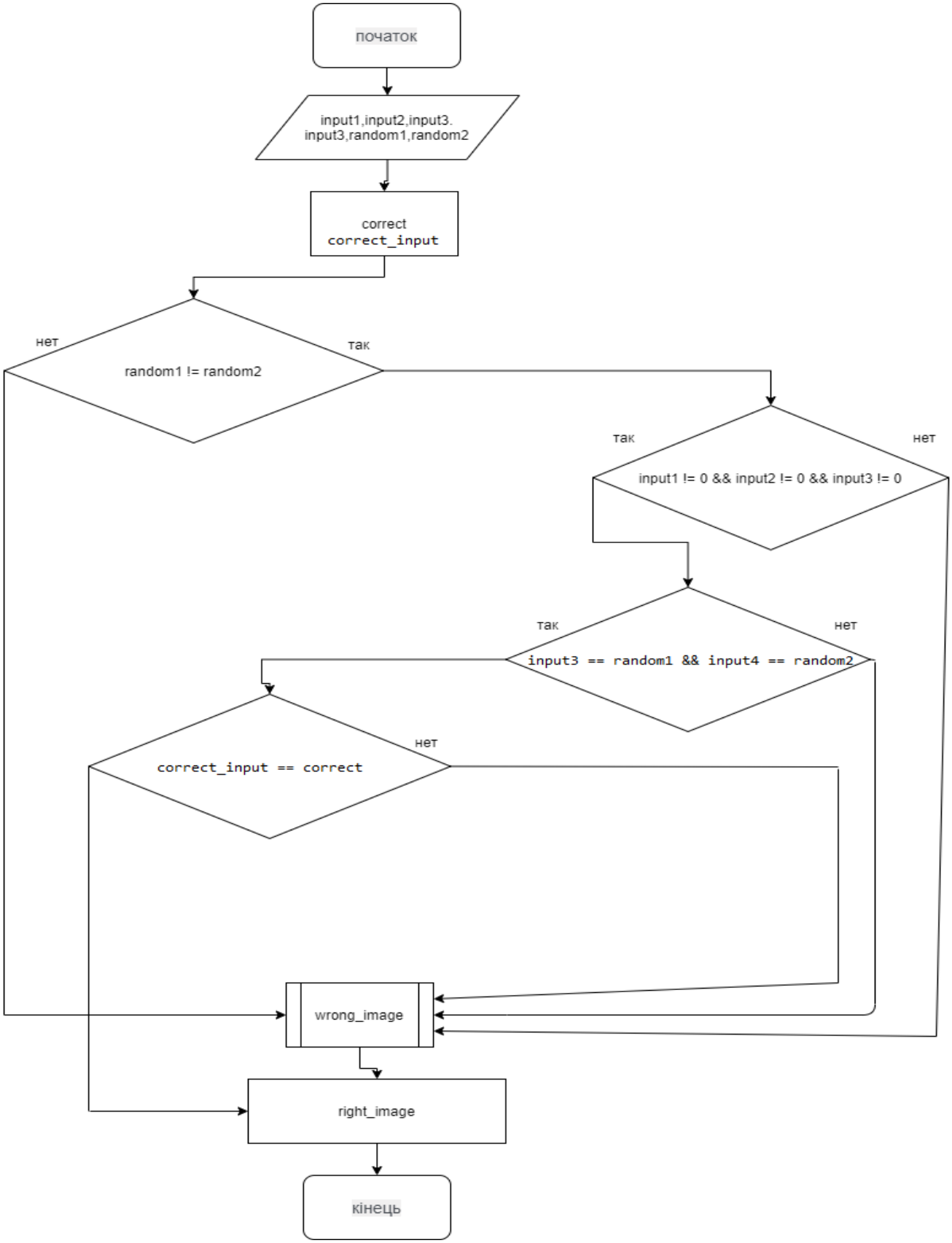
фрагмент коду ліва кнопка з рис 18.

```
1 reference
private void next_Click(object sender, EventArgs e)
{
    q1.Visible = true;anspanel.Visible = true;ans1.Visible = true;ans2.Visible = true;ans3.Visible = true;ans4.Visible = true;
    t++;if (t >= 8){
        t = 8;
    }
    label3.Text = Convert.ToString(t) + "/8";label_random4.Text = Convert.ToString(t);
    Random random1 = new Random();
    newrandom:
        int q1random1 = random1.Next(3, 15);
        int q1random2 = random1.Next(3, 15);
    if (q1random2 == q1random1)
    {
        goto newrandom;
    }
    label_random1.Text = Convert.ToString(q1random1);
    label_random2.Text = Convert.ToString(q1random2);
    var rnd = new Random();
    string[] answers;
    // code for loading questions !!
    answer1.Visible = true;

    switch (t)
    {
        case 1:
            //.show
            q1_input_1.Visible = true;
            q1_input_2.Visible = true;
            q1_2.Visible = true;
            q1_3.Visible = true;
            q1_4.Visible = true;
            inputp2.Visible = false;
            inputp3.Visible = false;
            this.label1.Text = "В классе учится " + q1random1 + " мальчиков и " + q1random2 + " девочек. Сколькими способами можно назначить одного дежурного?";
            this.ans1.Text = "Правило произведения";
            this.ans2.Text = "Правила сложения";
            this.ans3.Text = "Размещения без повторений";
            this.ans4.Text = "Правило произведения";
            label5.Text = "Правила сложения";
            break;
        case 2:
            this.ans1.Text = "Правило произведения";
            this.ans2.Text = "Правила сложения";
            this.ans3.Text = "Размещения без повторений";
            this.ans4.Text = "Правило произведения";
            label5.Text = "Правило произведения";
            inputp1.Visible = false;
        //inputp2.Visible = true;
        this.label1.Text = "В классе учится  " + q1random1 + " мальчиков и " + q1random2 + " девочек. Сколькими способами можно назначить двух дежурных?";
        label_random1.Text = Convert.ToString(q1random1);
        label_random2.Text = Convert.ToString(q1random2);
        break;
    case 3:
        this.ans1.Text = "Сочетания без повторений";
        this.ans2.Text = "Правила сложения";
        this.ans3.Text = "Размещения без повторений";
        this.ans4.Text = "Правило произведения";
        label5.Text = "Сочетания без повторений";
        inputp2.Visible = false;
        if (q1random2 >= q1random1)
        {
            this.label1.Text = "Необходимо выбрать в подарок  " + q1random1 + " из " + q1random2 + " имеющихся различных книг. Сколькими способами можно это сделать?";
            this.label_random1.Text = Convert.ToString(q1random2);
            this.label_random2.Text = Convert.ToString(q1random1);
        }
        else
        {
            this.label1.Text = "Необходимо выбрать в подарок  " + q1random2 + " из " + q1random1 + " имеющихся различных книг. Сколькими способами можно это сделать?";
            this.label_random1.Text = Convert.ToString(q1random1);
            this.label_random2.Text = Convert.ToString(q1random2);
        }
        break;
    case 4:
        this.ans1.Text = "Сочетания без повторений";
        this.ans2.Text = "Сочетания с повторениями";
        this.ans3.Text = "Размещения без повторений";
        this.ans4.Text = "Правило произведения";
        label5.Text = "Сочетания с повторениями";
        inputp3.Visible = false;
        if (q1random2 >= q1random1)
        {
            this.label1.Text = "В кондитерском магазине продавались  " + q1random1 + " сорта пирожных: наполеоны, эклеры, песочные и слоеные. Сколькими способами можно купить " + q1random2 + " пирожных?";
            this.label_random1.Text = Convert.ToString(q1random2);
            this.label_random2.Text = Convert.ToString(q1random1);

        }
        else
        {
            this.label1.Text = "В кондитерском магазине продавались  " + q1random2 + " сорта пирожных: наполеоны, эклеры, песочные и слоеные. Сколькими способами можно купить " + q1random1 + " пирожных?";
            this.label_random1.Text = Convert.ToString(q1random1);
            this.label_random2.Text = Convert.ToString(q1random2);
        }
        break;
    case 5:
```

рис.18

ДОДАТОК Б (фрагмент коду та його блок схеми)

```
            //Q4
            // vars
            var input1 = Convert.ToInt32(q4_input_1.Text);
            var input2 = Convert.ToInt32(q4_input_2.Text);
            var input3 = Convert.ToInt32(q4_input_3.Text);
            var input4 = Convert.ToInt32(q4_input_4.Text);
            var random1 = Convert.ToInt32(label_random1.Text);
            var random2 = Convert.ToInt32(label_random2.Text);
            // this is for founds correct answer
            correct = Factorial.factorial_Recursion(random1 + random2 - 1) /
(Factorial.factorial_Recursion(random1) * Factorial.factorial_Recursion(random2 - 1));
            if (random1 != random2)
            {
                if (input1 != input2 && input1 != 0 && input2 != 0 && input3 != 0 && input4 !=
0)
                {
                    if (input1 + input2 == random1 + random2)
                    {
                        if (input3 == random1 && input4 == random2)
                        {
                            correct_input = Factorial.factorial_Recursion(input1 + input2 - 1)
/ (Factorial.factorial_Recursion(input3) * Factorial.factorial_Recursion(input4 - 1));
                            this.answer_4.Text = Convert.ToString(correct_input);
                            if (correct_input == correct)
                            {
                                right_image.Visible = true;
                                System.Threading.Tasks.Task.Delay(1000).ContinueWith(_ => {
Invoke(new MethodInvoker(() => { right_image.Visible = false; })); });
                            }
                            else
                            {
                                wrong_image.Visible = true;
                                System.Threading.Tasks.Task.Delay(1000).ContinueWith(_ => {
Invoke(new MethodInvoker(() => { wrong_image.Visible = false; })); });
                            }
                        }
                    }
                    else
                    {
                        wrong_image.Visible = true;
                        System.Threading.Tasks.Task.Delay(1000).ContinueWith(_ => { Invoke(new
MethodInvoker(() => { wrong_image.Visible = false; })); });
                    }
                }
                else
                {
                    wrong_image.Visible = true;
                    System.Threading.Tasks.Task.Delay(1000).ContinueWith(_ => { Invoke(new
MethodInvoker(() => { wrong_image.Visible = false; })); });
                }
            }
```


початок
input1,input2,input3.
input3,random1,random2
correct
correct_input
нет
random1 != random2
так
так
input1 != 0 && input2 != 0 && input3 != 0
нет
так
input3 == random1 && input4 == random2
нет
correct_input == correct
нет
wrong_image
right_image
кінець

ДОДАТОК **С** (Електронні матеріали (диск))

ДОДАТОК Д

Зразок оформлення переліку умовних позначень, символів, одиниць, скорочень, термінів

Умовні позначення, символи, скорочення, терміни	*Пояснення умовних позначень, скорочень, симовлів*
int	*целочисленный тип данных.*
float	*тип данных с плавающей запятой.*
double	*тип данных с плавающей запятой двойной точности.*
char	*символьный тип данных.*
bool	*логический тип данных.*
class Класи	**Класи** є основним типом в мові ***C#***. Клас являє собою структуру даних, яка об'єднує в собі значення (поля) і дії (методи і інші функції-члени). Клас надає визначення для динамічно створюваних екземплярів класу, які також іменуються об'єктами. Класи підтримують механізми успадкування та поліморфізму, які дозволяють створювати похідні класи, що розширюють і уточнюють визначення базових класів.
Конструкторы	кожен раз, коли створюється клас або структура, викликається конструктор. Клас або структура може мати кілька конструкторів, які приймають різні аргументи. Конструктори дозволяють програмісту задавати значення за замовчуванням, обмежувати число установок і писати код, який є гнучким і зручним для читання.
Об'єкт	замкнуті, логічно несуперечливі, завжди коректні дані з чітко визначеним універсальним інтерфейсом доступу до них.
Циклом	називається блок коду, який для виконання завдання потрібно повторити кілька разів. Кожен цикл складається з блоку перевірки умови повторення циклу тіла циклу Цикл виконується до тих пір, поки блок перевірки умови повертає істинне значення. тіла циклу Цикл виконується до тих пір, поки блок перевірки умови

	повертає істинне значення. Тіло циклу містить послідовність операцій, яка виконується в разі істинного умови повторення циклу. Після виконання останньої операції тіла циклу знову виконуєтьс в разі істинного умови повторення циклу. Після виконання останньої операції тіла циклу знову виконується операція перевірки умови повторення циклу. Якщо ця умова не виконується, то буде виконана операція, що стоїть безпосередньо після циклу в коді програми.
Оператор if	Для организации условного ветвления язык C# унаследовал от C и C++ конструкцию if...else. Ее синтаксис должен быть интуитивно понятен для любого, кто программировал на процедурных языках: if (умова) оператор (оператори) else оператор (оператори) Якщо по кожному з умов потрібно виконати більше одного оператора, ці оператори повинні бути об'єднані в блок з допомогою фігурних дужок {...}. (Це також стосується інших конструкцій C #, в яких оператори можуть бути об'єднані в блок - таких як цикли for і while.) Варто звернути увагу, що на відміну від мов C і C ++, в C # умовний оператор if може працювати тільки з Булевського виразами, але не з будь значеннями на кшталт -1 і 0.В операторі if можуть застосовуватися складні вирази, і він може містити оператори else, забезпечуючи виконання більш складних перевірок. Синтаксис схожий на застосовуваний в аналогічних ситуаціях в мовах C (C ++) і Java. При побудові складних виразів в C # використовується цілком очікуваний набір логічних операторів.
Змінна	це «осередок» оперативної пам'яті комп'ютера, в якій може зберігатися будь-яка інформація. У програмуванні змінна, як і в математиці, може мати назву, що складається з однієї латинської букви, але також може складатися з декількох символів, цілого слова або кількох слів.
Факторіал	функція, визначена на множини невід'ємних цілих чисел. Назва походить від лат. factorialis - діючий, що

	виробляє, умножающий; позначається n !, вимовляється ен факторіал.
***оператор* switch**	Другим оператором вибору в C # є оператор switch, який забезпечує багатонаправлені розгалуження програми. Отже, цей оператор дозволяє зробити вибір серед кількох альтернативних варіантів подальшого виконання програми. Незважаючи на те що у багатьох напрямах перевірка може бути організована за допомогою послідовного ряду вкладених операторів if, в багатьох випадках більш ефективним виявляється застосування оператора switch. Цей оператор діє таким чином. Значення виразу послідовно порівнюється з константами вибору із заданого списку. Як тільки буде виявлено збіг з одним з умов вибору, виконується послідовність операторів, пов'язаних з цією умовою.
Змінна	це «осередок» оперативної пам'яті комп'ютера, в якій може зберігатися будь-яка інформація. У програмуванні змінна, як і в математиці, може мати назву, що складається з однієї латинської букви, але також може складатися з декількох символів, цілого слова або кількох слів.
Лейбли case	Перший вид лейбла - це case (або просто «кейс»), який оголошується з використанням ключового слова case і має константний вираз. Константне вираз - це те, яке виробляє константне значення, іншими словами: або літерал (наприклад, 5), або перерахування (наприклад, COLOR_RED), або константу (наприклад, змінна x, яка була оголошена з ключовим словом const). Константне вираз, що знаходиться після ключового слова case, перевіряється на рівність з виразом, що знаходяться біля ключового слова switch. Якщо вони збігаються, то тоді виконується код під відповідним case-му.
Button	Елемент управління Windows Forms **Button** дозволяє користувачеві клацнути його для виконання дії. На елементі управління **Button** можуть відображатися текст і зображення. При натисканні кнопки мишею елемент управління виглядає так, як ніби його натискають і відпускають.

Panel	Windows Forms елементи управління **Panel** використовуються для надання ідентифікованого групування для інших елементів управління. Як правило, панелі використовуються для поділу форми на функції. Елемент управління Panel аналогічний елементу управління GroupBox. Проте, тільки елемент управління **Panel** може мати смуги прокрутки, а тільки елемент управління GroupBox відображає заголовок.
PictureBox	Елемент управління Windows Forms **PictureBox** використовується для виведення зображень в форматі точкових малюнків, GIF, JPEG, метафайлів і значків.
Toolbox	Вікно **Toolbox** має цікаву властивість: ви можете скопіювати фрагмент коду в нього, клацнувши на області коду і перетягнувши її в вікно **Toolbox**. Ви можете також перейменувати і змінити таким чином ваші фрагменти коду, зробивши їх дійсно корисними для презентацій або зберігання часто використовуваних фрагментів. Вікно **Toolbox** містить всі доступні компоненти для активного в даний момент документа, відкритого в головному робочому вікні. Ними можуть бути візуальні компоненти, наприклад кнопки і текстові поля; невидимі, сервісні об'єкти, наприклад таймери або реєстраційні журнали системних подій; і навіть елементи дизайну, такі як класи та інтерфейсні об'єкти, що використовуються в інструменті Class Designer.