Форма № 7

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра механіки, автоматизації та інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Створення програмного інструментарія викладача розділу «Статика»»
(тема кваліфікаційної роботи українською мовою)
«Creation of software tools for the teacher of the «Statics» section»
(тема кваліфікаційної роботи англійською мовою)

Виконала: здобувачка денної/заочної форми навчання
спеціальності 126 – Інформаційні системи та технології
(код, назва спеціальності)
Освітня програма «Інформаційні системи та технології»
(назва)
Муленсон Єва Віталіївна
(прізвище, ім'я, по-батькові здобувача)

Керівник К.ф.-м.н., доц.Рачинська.А.Л.
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент ст.викл. Косирева Л.А.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту: Протокол засідання кафедри _____ № ___ від ____.____. 20___ р. Завідувач(ка) кафедри _____ Алла РАЧИНСЬКА (підпис)	Захищено на засіданні ЕК № _____ протокол № __від ____.____.20___ р. Оцінка_____/_____/_____ (за національною шкалою/шкалою ECTS/ бали) Голова ЕК _____ Володимир ВИЧУЖАНІН (підпис)

Одеса 2023

АНОТАЦІЯ

Цей диплом спрямований на розробку програмного набору інструментів, спеціально призначених для допомоги викладачам, які викладають предмет "Статика". Робота охоплює кілька важливих аспектів, таких як системи матеріальних точок, зв'язки між ними, координація систем сил, реакції на зв'язки та інше.

Під час нашого дослідження щодо створення цього набору інструментів було досягнуто кілька важливих завдань:

1) Визначення вимог викладача: ми працювали над визначенням функцій, які дозволять розробити програму із зрозумілим інтерфейсом, який потрібен користувачам під час використання інструментів при викладанні предмету "Статика", таких як гнучкість інтерфейсу користувача або глибина функціональності.

2) Функціональні можливості набору інструментів: враховуючи думки та пропозиції від цільових користувачів, ми намагалися визначити можливості, які дозволять ефективно викладати предмет "Статика" викладачам.

3) Аналіз поточних рішень та інструментів: було проведено аналіз існуючих інструментів, що використовуються при викладанні предмету «Статика». Аналізуючи їх переваги та недоліки, ми змогли створити надійну основу для нашого проекту.

4) Концептуалізація та технічний розвиток:технічна експертиза, спільно з уважним врахуванням вимог користувачів дозволили створити концепцію, необхідну для створення власного набору інструментів, який підходить для викладачів, що викладають предмет «Статика».

5) Оцінка ефективності: нарешті, ми оцінили ефективність програми, враховуючи важливі критерії та показники продуктивності, такі як рівень задоволеності та покращення якості навчання студентів.

ABSTRACTS.

The focal point of this thesis is on developing a software toolkit tailored towards assisting teachers who teach the subject matter included in «Statics». This work encompasses several crucial aspects such as material points systems, links between them, force systems coordination, link reactions etcetera.

During our research on creating this toolkit several essential tasks achieved include;

1) Identification of Teacher specifications: we worked towards identifying what features are critical to enable scalability along with critical interfaces that user operatives require when using toolkits while teaching Statics subjects such as user interface flexibility or functionality depth.

2) Functional capabilities of the toolkit: taking into account over several iterations from targeted users, we looked towards identifying the critical capabilities that enable effective teaching of Statics subjects for teachers.

3) Analysis of current solutions and tools: to bolster our research process and provide relevant comparison metrics, we designed a thorough analysis segment focused on researching existing tools used in the teaching of Static subjects. By analyzing their strengths and weaknesses we were able to forge a reliable framework for our tool-kit design.

4) Conceptualization and Technical Development: technical expertise, alongside careful consideration to user requirements derived from phase 1 and 2 allowed us to conceive the blueprint essential in setting-up a custom toolkit suitable for educators teaching statics subjects.

5) Evaluation of effectiveness: lastly, we evaluated the effectiveness of our software toolkit by considering vital measurement criteria distinctively tailored towards measuring levels such as usability scores among teachers using this tool kits. We also considered other performance indicators like; satisfaction rates along with quality learning improvement in students.

ЗМІСТ

Стор.

ВСТУП 5
1 ДОСЛІДЖЕННЯ СТАТИЧНОГО ТІЛА 7
1.1 Введення у кінематику 7
1.2 Методи дослідження статичного тіла 8
2 ЕЛЕМЕНТАРНА ГЕОМЕТРИЧНА СТАТИКА 10
2.1 Система матеріальних точок 10
2.2 Зв'язки 11
2.3 Координати системи 13
2.4 Сили відображаються 14
2.5 Реакції зв'язків 17
2.6 Формули 21
3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 24
3.1 Платформа для розробки 24
3.2 Використання поліморфізму у формі застосунку 29
3.3 Об`єктнозорієнтоване програмування 30
3.4 Реалізація класів 31
3.5 Додання формул до програми 36
4 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ 41
ВИСНОВКИ 45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
ДОДАТОК А 47

ВСТУП

В сучасному світі інформаційних технологій та швидкого розвитку комп'ютеризованого середовища особливо важливою є проблема покращення процесу навчання інженерних дисциплін, зокрема статики. Статика є однією з основних дисциплін, що досліджує рівновагу тіл та систем, і вона відіграє ключову роль у підготовці майбутніх інженерів різних спеціалізацій.

Однак, навчання статики може зіткнутись із проблемами розуміння завдання для багатьох студентів. Ця дисципліна вимагає розуміння складних математичних концепцій, вирішення складних задач та аналізу структурних систем. Багато студентів зіштовхуються з труднощами під час опанування цих концепцій і застосування їх на практиці. Відсутність зручних інструментів, що підтримують навчання статики, може стати причиною погіршення рівня освіти та зростання відсотку студентів, які відмовляються від інженерних спеціальностей через незрозумілість та складність матеріалу.

Проте, з розвитком комп'ютерних технологій та програмного забезпечення відкриваються нові можливості для покращення процесу навчання статики. Створення програмного інструментарію для викладачів розділу "Статика" може суттєво полегшити роботу викладачів і поліпшити процес навчання студентів. Такий програмний інструментарій має на меті розробку програми, яка здатна вирішувати різні статичні задачі шляхом застосування певних формул та алгоритмів.

Метою даної роботи є створення програмного інструментарію викладача розділу «Статика», який забезпечить зручність, ефективність та індивідуалізацію процесу навчання статики. Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Розробити програму, яка здатна вирішувати різні статичні задачі.

2. Забезпечити гнучкість програми, щоб вона могла вирішувати формули за разними заданими значеннями.

3. Розробити інтуїтивний інтерфейс користувача, який буде зрозумілим для викладачів з різним рівнем технічної підготовки.

4. Забезпечити можливість візуалізації схем та сил реакції, що дозволить студентам краще розуміти і аналізувати отримані відповіді.

Для досягнення поставленої в роботі мети будуть використані наступні вихідні матеріали і методи дослідження:

1. Вивчення наукової літератури та існуючих програмних рішень в галузі статики.

2. Аналіз потреб викладачів та студентів у зручному інструментарії для навчання статики.

3. Розробка та програмування програмного інструментарію, заснованого на відповідних алгоритмах та формулах.

4. Проведення тестування та оцінка ефективності розробленого інструментарію.

Об'єкт дослідження – рівновага довільної плоскої системи сил. На систему можуть діяти момент, сили та розподілене навантаження. Сили зв’язку залежать від типу опори.

Предмет дослідження – у даній роботі є сили реакції зв’язку опор різного типу для однакового навантаження.

Таким чином, створення програмного інструментарію викладача розділу «Статика» має значення для поліпшення процесу навчання статики, забезпечуючи зручність, ефективність та індивідуалізацію.

ДОСЛІДЖЕННЯ СТАТИЧНОГО ТІЛА

1.1 **Введення у кінематику**

У роботі [2] стверджується, що кінематика – це розділ фізики та механіки, який вивчає без участі сили, що викликають рух тіл. Ми вивчаємо рух тіл, включаючи положення, швидкість, прискорення та траєкторію, незалежно від причини руху.

Кінематика використовується в багатьох галузях науки і техніки, включаючи:

1. Механіка: Кінематика є основою для вивчення механіки руху тіла. Його можна використовувати для визначення положення, швидкості та прискорення об’єкта в просторі та часі, допомагаючи вивчати закони руху та прогнозувати його динаміку.

2. Робототехніка: кінематика використовується для моделювання та керування рухом робота. Його можна використовувати для обчислення траєкторій, визначення кінематичних обмежень і забезпечення точності та безпеки руху робота.

3. Автомобільна промисловість: кінематика використовується в автомобільному проектуванні для аналізу руху коліс, підвіски, рульового управління та інших механізмів. Це допомагає визначити оптимальні параметри транспортного засобу та забезпечує комфорт і безпеку пасажирів.

4. Аерокосмічна промисловість: аерокосмічна техніка використовує кінематику для моделювання траєкторій ракет, супутників та інших космічних апаратів. Це дозволяє визначити швидкості, траєкторії та кутові швидкості, необхідні для досягнення бажаної траєкторії. Поки що ми надали деяку загальну інформацію про використання кінематики. Однак, заглиблюючись у тему статики, кінематику можна пов’язати з аналізом руху тіл у статичних ситуаціях. Це означає, що кінематика може допомогти

визначити різні параметри, пов’язані з положенням тіла, навіть коли система перебуває в рівновазі (без руху).

Кінематика - це розділ механіки, який вивчає рух об'єкта без урахування сил, які викликають цей рух. Створюючи програмний засіб для вчителів у розділі Статика, кінематику можна пов’язати з процесом моделювання та візуалізації рухів тіла. Він зосереджується на геометричних властивостях об’єктів і використанні ними статики та кінематики як незалежних факторів для визначення їхнього положення в просторі.

1.2 Методи дослідження статичного тіла

Існує кілька методів дослідження статичного тіла, які дозволяють аналізувати його рівновагу, взаємодію з оточенням та інші характеристики. Ось декілька загальноприйнятих методів:

1. Аналітичний метод: Цей метод використовує математичні рівняння та принципи механіки для аналізу статичного тіла. Він базується на застосуванні законів рівноваги та рівнянь механіки для визначення сил, моментів і реакцій опор. Цей метод дозволяє точно обчислити параметри статики тіла за умови, що відомі всі необхідні дані.

2. Експериментальний метод: Цей метод полягає в проведенні фізичних експериментів для визначення різних параметрів статичного тіла. Він включає в себе вимірювання сил, моментів та реакцій опор, використовуючи спеціальні прилади, такі як датчики сили, навантажувачі та інші пристрої. Цей метод може бути корисним, коли точні математичні моделі складні або недоступні.

3. Комп'ютерне моделювання: За допомогою комп'ютерного програмування та спеціалізованих програмних засобів можна створити віртуальну модель статичного тіла та аналізувати його поведінку. Комп'ютерні моделі дозволяють проводити чисельні розрахунки, симулювати

різні умови та виконувати оптимізацію конструкцій. Цей метод дозволяє швидко та ефективно виконувати аналіз статики тіла з високою точністю.

Ці методи можуть використовуватися окремо або в комбінації для дослідження статичного тіла.

Для дослідження статичного тіла можна застосовувати різні методи, залежно від його складності, доступності даних та мети дослідження. Аналітичний метод базується на математичних рівняннях і законах рівноваги. Експериментальний метод використовує фізичні експерименти та прилади для вимірювання сил та реакцій опор. Комп'ютерне моделювання залучає віртуальну модель тіла та чисельні розрахунки. Вибір методу залежить від складності тіла, наявності даних та ресурсів, а також від мети дослідження, будь то отримання точних значень або аналіз впливу та оптимізація конструкції.

ЕЛЕМЕНТАРНА ГЕОМЕТРИЧНА СТАТИКА

2.1 Система матеріальних точок

В роботі [1] описано що сукупність (множина) матеріальних точок (частинок) називається системою матеріальних точок (частинок). Таку систему можна утворити з будь-якої множини матеріальних точок, вибраних абсолютно випадково; тому кожна точка може або належати до розглянутої системи, або не належати. Якщо система матеріальних точок має властивість, що рух кожної точки залежить від положення і руху інших точок системи, то така система називається *механічною системою* матеріальних точок. Отже, для того, щоб система була механічною, необхідно, щоб точки системи були яким-небудь чином пов'язані між собою; при цьому між точками системи будуть діяти сильні взаємодії (наприклад, між планетами сонячної системи, якщо їх розглядати як матеріальні точки). Будь-яке матеріальне тіло (тверде, рідке або газоподібне) представляє собою механічну систему, що складається з дуже великої кількості матеріальних частинок (точок), пов'язаних між собою силами інтрамолекулярного діяння, які накладають певні обмеження на взаємні відстані між частинками залежно від природи тіла. Будь-яка сукупність матереальні тіла, так або інакше пов'язані між собою, також утворюють механічну систему (ферму, механізм, машину і т. д.).

Якщо точки системи або тіла пов'язані незмінно, тобто так, що взаємна відстань між будь-якими двома точками залишається постійною, то така *система* називається *незмінною системою*, а тіло - *абсолютно твердим тілом*; у протилежному випадку система називається змінною, а тіло - *деформованим*.

Положення системи: визначається, якщо відомо положення кожної з точок, що складають систему, і навпаки; так само рух системи відомий, якщо відомий рух кожної точки, що належить до системи, і навпаки.

2.2 Зв'язки

Робота [2], свідчить про те, що якщо кожна з точок системи може займати довільне положення в просторі і мати довільну швидкість, то система називається *вільною*; у протилежному випадку система буде *невільною*. Умови, що накладають обмеження на рух системи, називаються *зв'язками*. Якщо зв'язок накладає обмеження лише на положення системи або на відносне положення точок, що складають систему, у тому сенсі, що система і її елементи не можуть займати довільне положення в просторі, то такий зв'язок називається *геометричним*; якщо ж зв'язок, крім того, накладає обмеження ще й на кінематичні характеристики (наприклад, на швидкості), то такий зв'язок називається *кінематичним*.

Геометричні зв'язки представляються аналітично рівняннями, які визначають залежність між координатами точок системи. Для системи, що складається з n матеріальних точок, положення яких визначається їх декартовими координатами $x_1, y_1, z_1, x_2, y_2, z_2, \dots$, рівняння геометричного зв’язку має вигляд:

$$f(x_1, y_1, z_1, x_2, y_2, z_2, \dots x_n, y_n, z_n) = 0 \qquad (2.1)$$

Чи скорочено:

$$f(x, y, z) = 0 \qquad (2.2)$$

Примітка. Відсутність індексів у виразі 2.2 означає тут і далі, що при x, y, z, потрібно мати на увазі індекси від 1 до n.

Рівняння кінематичного зв’язку накладає обмежень не тільки на положення, але і на швидкість точок системи, має вигляд:

$$f(x, y, z, \dot{x}, \dot{y}, \dot{z}) = 0 \qquad (2.3)$$

Зв'язки зазвичай реалізуються у вигляді різних об'єктів, які обмежують свободу переміщення точок системи. Якщо вплив зв'язку не може бути припинений або, іншими словами, система не може звільнитися від зв'язку, то такий зв'язок називається незвільненим; якщо система може покинути зв'язок, то зв'язок називається *звільненим*.

Зв'язки, виражені рівняннями у вигляді 2.1 або 2.3, є незвільненими. Якщо зв'язок є звільненим (і геометричним), то він виражається нерівністю у вигляді:

$$f(x, y, z) \geq 0 \ \text{ чи } \ f(x, y, z) \leq 0 \qquad (2.4)$$

Зв'язки, які не залежать від часу, називаються *стаціонарними* або *склерономними* (за термінологією Больцмана). Якщо ж зв'язок залежить від часу, то він називається *нестаціонарним* або р*еономним* зв'язком. Наприклад, нерухома поверхня або крива, по якій примушена рухатися точка, будуть склерономним зв'язком; якщо ж ця поверхня або крива рухаються, то зв'язок буде реономним.

Зв'язки, виражені рівняннями у вигляді 3.1, 3.3 або 3.4, є склерономними. Рівняння реономного зв'язку (і, зокрема, геометричного, незвільненого) має вигляд:

$$f(x, y, z, t) = 0 \qquad (2.5)$$

Варто зазначити, нарешті, що зв'язки відносно розглянутої системи можна розділити на внутрішні та зовнішні. Внутрішній зв'язок називається таким, який не перешкоджає переміщенню всієї системи в цілому, а обмежується лише відносним розташуванням точок системи; в іншому випадку зв'язок називається зовнішнім. Таким чином, якщо зв'язками

служать тіла, що належать до системи, то ці зв'язки будуть внутрішніми, і навпаки. Систему, яка має лише внутрішні зв'язки, і вид зв'язку, також називають вільною системою.

2.3 Координати системи

У роботі [3] сказано що незалежні між собою величини, що визначають положення або *конфігурацію* системи матеріальних точок відносно деякої системи відліку, називаються *координатами системи*. Конфігурацію системи ми можемо геометрично зобразити точкою простору, число вимірів якого дорівнює числу координат системи. Якщо на систему накладено лише геометричні зв'язки, то число координат системи називається *кількістю ступенів свободи* цієї системи.

Нехай ми маємо систему, що складається з n матеріальних точок. Положення кожної точки визначається трьома декартовими координатами (x_v, y_v, z_v), де v - номер точки; отже, положення всієї системи, якщо на неї не накладено зв'язків, буде визначатися $3n$ координатами: $x_v, y_v, z_v \quad (v = 1,2,\dots,n)$, та така система буде мати $3n$ ступенів свободи.

Припустимо тепер, що система підлегла k геометричним зв'язкам виду 3.1, тобто:

$$f_x(x_1, y_1, z_1, x_2, y_2, z_2, \dots x_n, y_n, z_n) = 0 \ (x = 1,2,\dots,k) \qquad (2.6)$$

Тоді $3n$ координат точок не будуть вже між собою незалежні і не можуть мати довільних значень, а будуть пов’язані k умовами 3.6; тому незалежних координат буде:

$$3n - k \qquad (2.7)$$

Таким чином, число координат системи, а отже і кількість ступенів свободи її буде $3n-k$.

За координати системи можемо прийняти в даному випадку будь які $3n-k$ декартові координати x_v, y_v, z_v , які будемо вважати незалежними, тоді інші k , з цих координат будуть функціями перших. Можна $3n-k$ незалежних декартових координат системи перетворити в інші за допомогою точкового перетворення висловивши їх у функціях $3n-k$ незалежних змінних $q_1, q_2, \dots q_{3n-k}$, тобто поклавши:

$$x_1 = x_1(q_1, q_2, \dots),\ y_1 = y_1(q_1, q_2, \dots) \text{ і так далі} \qquad (2.8)$$

Де $x_1, y_1, \dots$ будуть аналітичними функціями змінних q_i , при тому що якобіан: $\frac{\partial(x_1, y_1, \dots)}{\partial(q_1, q_2, \dots)} \neq 0$. Тоді $q_1, q_2, \dots q_{3n-k}$ будуть координатами системи в спільному випадку криволінійними. Декартовими координати точок системи у цьому випадку будуть функціями координат $q_1, q_2, \dots q_{3n-k}$, тобто:

$$x_1, y_1, z_1, x_2 \dots y_n, z_n | q_1, q_2, \dots q_{3n-k} \qquad (2.9)$$

2.4 Сили відображаються

Робота [4] свідчить про те, що матеріальні тіла можуть взаємодіяти одне з одним через прямий контакт або на відстані; залежно від цього, сили, що використовуються в механіці як міра взаємодії тіл, можна розділити на дві категорії.

1) *Поверхневі сили* - сили, що діють на точки поверхні тіла. Вони виникають при взаємодії одного тіла з іншим безпосереднім контактом і

прикладені до тієї частини поверхні тіла, де взаємодіючи тіла знаходяться в контакті одне з одним.

Нехай тіло 1 діє на тіло 2, торкаючись його вздовж деякої поверхні (рис. 2.1). Сили, що діють на цю поверхню, характеризуються їхнім напруженням p, тобто величиною сил,

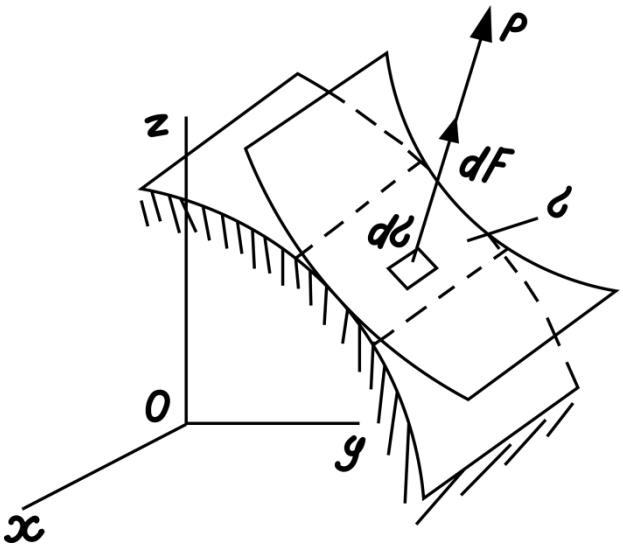

Рисунок 2.1 – Поверхневі сили

що припадає на одиницю площини. Якщо на елемент $d\sigma$ плозини контакту діє сила dF, то:

$$p = \frac{dF}{d\sigma} \tag{2.10}$$

Величина p буде загальною функцією координат точки, визначальною становище елемента $d\sigma$; розмірність p буде $\frac{сила}{(длина)^2}$.

Якщо на дуже малий елемент поверхні діє скінченна сила F, то, ігноруючи розмір цього елемента, ми можемо уявити абстрактне уявлення про скупчену силу, прикладену до тіла в одній точці. З достатньою точністю скупченою силою можна, наприклад, вважати силу, з якою діє нитка на тіло

(рис. 2.2), прикріплена до тіла в точці А (фактично, звичайно, нитка прикріплена не до «точки», а до якогось елемента поверхні тіла).

2) *Масові чи об'ємні* сили – сили, які діють на всі частинки тіла. До таких сил відносяться сили далекодії, такі, наприклад, як сили тяжіння або тяжіння. Масові сили характеризуються їх напругою f, тобто, величина сили, яка приходить на одиницю об'єму.

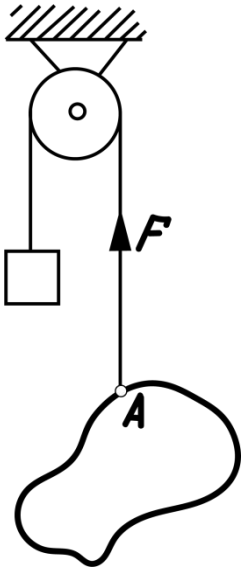

Рисунок 2.2 – Скупчена сила

При ваємодії тіл (1) та (2) на елемент об'єму $d\tau$ тіла 2 діє сила dF (рис. 2.3).

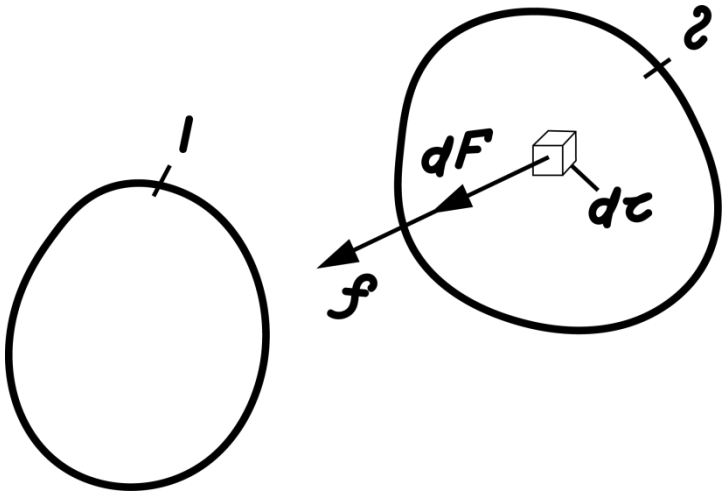

Рисунок 2.3 – Масові чи об'ємні сили

Тобто:

$$f = \frac{dF}{d\tau} \tag{2.11}$$

Величина f є взагалі функцією координат точки визначальною положення елемента об'єму $d\tau$. Розмірністю f буде $\frac{маса}{(длина)^3}$. Якщо прискорення, повідомляється елементу силою dF, позначимо w, то будемо мати:

$$dF = m \times w = \rho w d\tau \tag{2.12}$$

Порівнюя рівняння 2.9 та 2.11, отримаємо:

$$f = \rho w \tag{2.13}$$

Якщо силу dF будемо відносити не до одиниці об'єму, а до одиниці маси, то з рівняння (10) отримаємо:

$$\frac{dF}{dm} = w \tag{2.14}$$

Звідси видно, що сила, віднесена до одиниці маси в даній точці тіла, дорівнює прискоренню, отриманому цією точкою від діючих на тіло масових сил.

Крім поділу сил на поверхневі та масові, сили, що діють на систему, можна розділити на внутрішні та зовнішні. *Внутрішні* сили виникають від

взаємодії частинок (тіл), що належать системі; *зовнішні* сили виникають від дії тіл, що не належать системі.

2.5 Реакції зв'язків

У роботі [5] зв'язки, накладені на точки системи, обмежують свободу руху цих точок, відхиляючи їх рух від того, який вони мали б при дії тих самих сил, якби були вільними від зв'язків. Тому ми можемо вважати, що ефект дії зв'язків такий самий, як дія сил, тому *дію зв'язків можна замінити відповідними силами, які називаються реакціями зв'язків* (аксіома зв'язків).

Реакції зв'язків за своєю природою дещо відрізняються від усіх інших сил, що діють на систему, не є реакціями. Ця відмінність полягає в тому, що реакція зв'язку не повністю визначається самим зв'язком; її модуль, а іноді й напрямок, залежать також від інших сил, що діють на систему, та від руху системи (при відсутності активних сил і руху реакції взагалі не виникають). Модуль і напрямок кожної *активної* сили (а також їх залежність від часу, координат точки прикладання сил та швидкості) відомо заздалегідь і з інших прикладених до системи сил залежать. Крім того, активні сили, що діють на нерухому систему, можуть надати їй певний рух (отже, їх називають «активними»); реакції зв'язків не володіють такою властивістю, тому їх також називають *пасивними* силами.

Зв'язки зазвичай реалізуються у вигляді різних тіл, що обмежують свободу руху системи. У цих випадках реакція зв'язку представляє собою силу, прикладену в точці, де зв'язок контактує з тілом, при цьому напрям реакції співпадає з напрямом, в якому зв'язок перешкоджає переміщенню тіла. Якщо таких напрямів декілька, то напрямки реакцій, а також їхні напруження, визначаються залежно від активних сил, що діють на тіло, та руху самого тіла. Крім того, треба зауважити, що реакція зв’язку є дія зв’язку на тіло; при цьому тіло діє на зв’язок із силою, рівною реакції та її протилежної, за законом дії та протидії.

Нехай, наприклад, тверде тіло вагою *P* підвішене в нерухомій точці *O* на нерозтяжній нитці, яка прикріплена до точки *A* тіла (рис. 2.4, а). Нитка, що виконує функцію зв'язку, надає реакцію *T*, прикладену в точці А тіла і спрямовану вздовж нитки; числове значення цієї реакції дорівнює вазі тіла *P* у даному випадку, оскільки нитка діє на тіло силою *T*, а тіло діє на нитку силою *P*. У разі, коли важке тіло вагою Р, підвішене на нитці до нерухомої точки *O* (рис. 2.4, б), здійснює коливання (маятник), реакція нитки *T* буде, як і раніше, спрямована вздовж нитки, але її чисельна величина залежатиме не тільки від *P* , але і від кут φ, та кутової швидкості $\frac{d\varphi}{dt}$, тобто, від руху тіла.

Якщо зв'язком слугує нерухома гладка поверхня (рис. 2.5), то вона надає реакцію М, прикладену в точці дотику тіла до цієї поверхні і спрямовану вздовж нормалі до неї.

Гладка поверхня, яка нерухома та без спротиву, забезпечує певну реакцію на дію тіла. Ця реакція може бути використана для передачі сили, забезпечення руху або підтримки об'єкта на поверхні. За допомогою реакції М тіло відчуває підтримку та може виконувати різноманітні рухи з упевненістю та стабільністю.

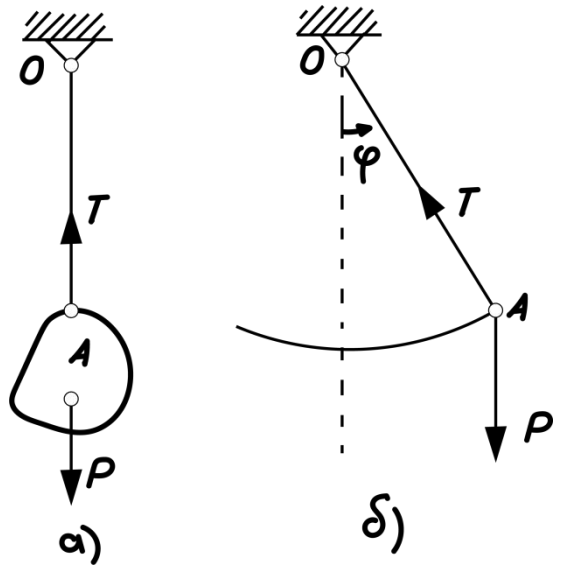

Рисунок 2.4 – Приклад твердого тіла підвішеного в нерухомій точці (а) та маятника (б)

Розуміння того, як реакція M діє на тіло і спрямована вздовж нормалі до поверхні, має важливе значення в механіці та інших галузях фізики. Воно дозволяє аналізувати та передбачати рухи тіла, визначати сили, що діють на нього, та враховувати вплив нерухомої гладкої поверхні. Це знання є основою для розвитку теорії та практичних застосувань у багатьох галузях, включаючи машинобудування, механіку, робототехніку та інші області.

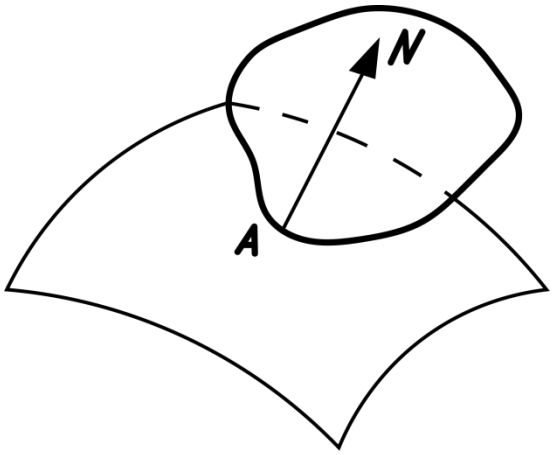

Рисунок 2.5 – Приклад де зв'язком слугує нерухома гладка поверхня

Напруження реакцій залежить від активних сил, що діють на тіло, і руху тіла. Нерухома гладка крива, що виконує функцію зв'язку (рис. 2.6), розвиває реакцію, прикладену в точці торкання *A* та спрямовану за нормаллю кривою.

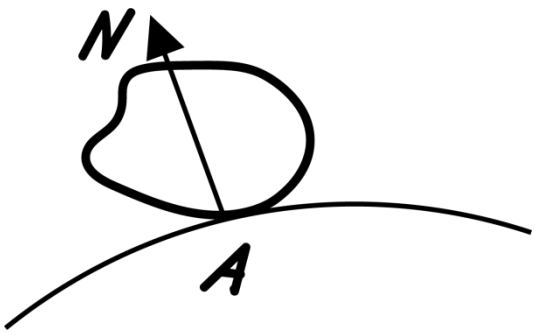

Рисунок 2.6 – Нерухома гладка крива виконує функцію зв'язку

У цьому випадку реакція може мати будь-яку напруженість та напрямок, що лежить у нормальній площині, проведеній до кривої через точку *A*. Якщо зв'язком є нерухома точка *O* (рис. 2.7), то вона може мати будь-яку за напруженістю та напрямком реакцію *M*, прикладену до цієї точки. Модуль та напрямок реакції у цих останніх двох випадках залежать від діючих активних сил та руху тіла.

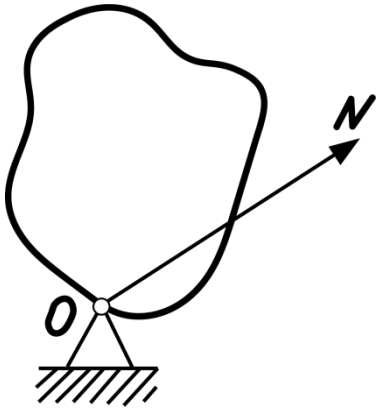

Рисунок 2.7 – Зв'язок з нерухомою точкою

Зв'язки, які розвивають нормальну реакцію, є ідеальними зв'язками (зв'язками без тертя), на відміну від зв'язків із тертям, які, крім нормальної реакції *M*, ще дають реакцію К, що лежить у касательній площині (рис. 2.8) і виникає через тертя. Внаслідок цього реакція зв'язку з тертям *M* може відхилятися від нормалі під певним кутом.

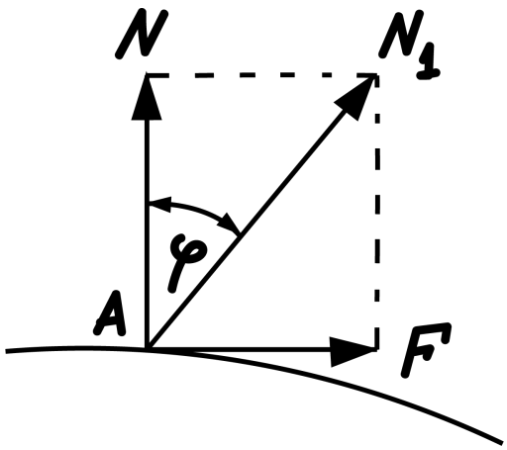

Рисунок 2.8 – Зв'язок із тертям

2.6 Формули

Визначення реакцій опор твердого тіла є важливим етапом у розв'язанні завдань з механіки та статики. Реакції опор є силами, які діють на тіло у точках контакту з підкладиною або опорами. Ці реакції забезпечують рівновагу та стійкість тіла під час дії зовнішніх сил.

Для визначення реакцій опор використовуються схеми та формули, які враховують геометричні характеристики тіла, розташування опор, прикладені сили та умови рівноваги. Це дозволяє встановити значення реакцій у певних точках та визначити їх напрям та силу.

Розберемо на прикладі першої схеми (рисунок 2.9) її формули.

Завданням є визначити реакції опор для того способу закріплення, при якому момент в закладенні має найменше числове значення.

Розглянемо систему сил, що врівноважуються, прикладених до конструкції. Дія зв'язків на конструкцію замінюємо їх реакціями (рис 2.9).

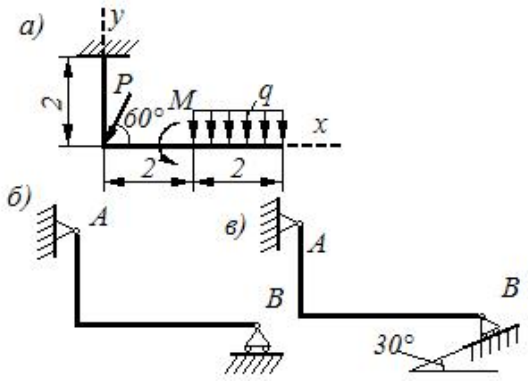

Рисунок 2.9 – Перша схема

Щоб з'ясувати, у якому разі момент у закладенні є найменшим, треба знайти його для всіх трьох схем.

Для частини А):

$$x: x_a - P\cos 60 = 0 \rightarrow x_a = P\cos 60 = 0{,}5P$$
$$y: y_a - P\cos 60 - 2q = 0 \rightarrow y_a = P\cos 60 + 2q \rightarrow$$
$$y_a = \frac{P\sqrt{3}}{2} + 2q = 0{,}5P\sqrt{3} + 2q \qquad (2.15)$$
$$M: M_a - M - P\cos 60 * 2 - q * 2 * 2 = 0$$
$$M_a = M + P + 4q;\ R_a = \sqrt{{x_a}^2 + {y_a}^2}$$

Для частини Б):

$$x: x_a - P\cos 60 = 0 \rightarrow x_a = 0{,}5P$$
$$y: y_a + y_b - P\cos 60 - 2q = 0$$
$$M: y_b * 4 - M - P\cos 60 * 2 - q * 2 * 2 = 0 \qquad (2.16)$$
$$y_b = 0{,}25(M + P + 4q)$$
$$y_a = 0{,}5P\sqrt{3} + 2q - y_b$$
$$R_a = \sqrt{{x_a}^2 + {y_a}^2};\ R_b = y_b;\ M_a = M_b = 0$$

Для частини В):

$$x: x_a - R_b\cos 60 - P\cos 60 = 0$$
$$y: y_a + R_b\sin 60 - P\sin 60 - 2q = 0$$
$$M: -R_b\cos 60 * 2 + R_b\sin 60 * 4 - P\cos 60 - q * 2 * 2 = 0$$
$$R_b(4\sin 60 - 2\cos 60) = P + 4q \qquad (2.16)$$
$$R_b = \frac{P + 4q}{2\sqrt{3} - 1}$$

$$x_a = 0{,}25R_b + 0{,}5P$$

$$y_a = 0{,}5P\sqrt{3} + 2q - 0{,}5\sqrt{3}R_b$$

$$R_a = \sqrt{{x_a}^2 + {y_a}^2}\ ;\ M_a = M_b = 0$$

ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

В процесі роботи зі створення програмного інструментарію викладача розділу «Статика» необхідно враховувати принципи та методи проектування системи, що забезпечать його ефективність та функціональність. Проектування системи передбачає створення відповідної архітектури, визначення функціональних вимог, вибір технологій та інструментів, розробку інтерфейсу користувача та інші важливі етапи. Давайте розглянемо проектування системи для створення програмного інструментарію викладача розділу «Статика».

1. Визначення вимог системи:

А) Встановлення функціональних вимог до програмного інструментарію, таких як можливість вирішення певного набору статичних задач на основі заданих формул, алгоритмів та вхідних даних.

Б) Визначення нефункціональних вимог, таких як швидкодія, зручність інтерфейсу тощо.

2. Аналіз і проектування:

А) Визначення основних компонентів системи, їх взаємозв'язок та взаємодію.

Б) Вибір технологій і програмних засобів, які найкраще підходять для реалізації системи.

3.1 Платформа для розробки

У роботі [6] описано що щоб обрати оптимальну платформу для розробки програмного інструментарію викладача розділу «Статика» важливо досліджуватись декількох кроків для забезпечення ефективності, зручності та успішного завершення проекту.

У роботі [7] описано кілька кроків, які допоможуть правильно обрати платформу:

1. Аналіз потреб і вимог проекту: Треба почати з аналізу потреб і вимог до проекту. Чітко визначити, які функції і можливості повинна мати програма, які інтеграції з іншими системами або інструментами є необхідними. Розробити список вимог, який буде слугувати основою для вибору платформи.

2. Дослідження доступних платформ: Провести дослідження різних платформ, які підходять для розробки програмного інструментарію. Розглянути їх особливості, функціональність, підтримку мов програмування та розширень, спільноту розробників, наявність документації та інші фактори, які можуть вплинути на проект.

3. Відповідність потребам проекту: Порівняти вимоги проекту з можливостями кожної платформи. Визначити, яка платформа найкраще задовольняє потреби щодо функціональності, інструментів, швидкості розробки та інших критеріїв, які вам важливі.

4. Врахування досвіду та навичок: При виборі платформи треба враховувати власний досвід і навички. Якщо є досвід роботи з конкретною платформою, це може зекономити час і зусилля, оскільки легше працювати вже зі знайомими з їх інструментами та можливостями. Треба врахувати також навички команди розробників, які будуть працювати над проектом.

Після аналізу та дослідження доступних платформ, для вирішення даної проблеми було обрано Visual Studio як основні інструменти для розробки програмного інструментарію викладача розділу «Статика». Це відмінний вибір, оскільки ця платформа має свої переваги та можливості, які можуть сприяти успішній реалізації проекту. Ось деякі переваг:

1. Інтегроване середовище розробки : Visual Studio надає потужне та зручне середовище для програмування. Вона має багатий функціонал, включаючи відлагодження, авто-доповнення коду, вбудовану підтримку версій контролю та інші інструменти, що полегшують процес розробки.

2. Підтримка .NET Framework: Visual Studio відмінно інтегрується з .NET Framework, що забезпечує широкі можливості для розробки програмного забезпечення на мові програмування C# або VB.NET. .NET Framework має вбудовану підтримку для роботи з базами даних, мережевими операціями та іншими технологіями, що можуть бути корисними для проекту.

3. Windows Forms: Ця технологія розробки графічного інтерфейсу користувача в середовищі .NET Framework дозволяє швидко створювати віконні додатки з використанням стандартних елементів управління. Windows Forms надає широкі можливості для створення інтерактивного та зручного інтерфейсу для користувача.

Програму було розроблено на мові C# (C Sharp), яка є мовою програмування, розробленою компанією Microsoft, і вона має свої плюси і мінуси. Давайте розглянемо деякі з них:

Плюси C#:

Об'єктозорієнтований підхід: C# підтримує об'єктно-орієнтоване програмування, що сприяє модульності, повторному використанню коду та легкості підтримки та розширення програм.

Інтегрованість з .NET Framework: C# використовує .NET Framework, що надає широкий набір бібліотек і функцій для розробки програмного забезпечення. Це спрощує розробку, забезпечує багатофункціональність і підтримку різних платформ.

Безпека: C# надає вбудовану підтримку для безпеки типів, управління пам'яттю та винятків, що допомагає запобігти багатьом типовим помилкам програмістів, таким як некоректне використання пам'яті або неправильні типи даних.

Підтримка мультимедіа: C# має потужні бібліотеки для роботи з мультимедіа, що дозволяє легко реалізовувати функціональність, пов'язану з обробкою зображень, аудіо та відео.

Мінуси C#:

Залежність від платформи Microsoft: C# і .NET Framework розроблені Microsoft, тому вони найбільш підходять для розробки на платформах Windows. Якщо вам потрібно розробляти кросплатформенні програми, можуть виникнути обмеження і складнощі.

Вимоги до ресурсів: Для запуску програм, написаних на C#, потрібна встановлена .NET Framework, що може становити додаткове навантаження для кінцевих користувачів, особливо якщо вони не мають попередньої встановленої версії.

Швидкодія: Хоча C# є мовою високого рівня, вона може бути трохи повільнішою в порівнянні з деякими низькорівневими мовами, такими як C++. Це може бути проблемою, якщо потрібна висока швидкодія для виконання обчислювально інтенсивних завдань.

Варто зазначити, що багато з недоліків C# можна зменшити або уникнути за допомогою правильного проектування та оптимізації коду. Остаточний вибір мови програмування залежить від конкретних потреб проекту та здатностей команди розробників.

На початковому етапі навчання було ознайомлено з можливостями об'єктозорієнтованого програмування. Досліджувалось створення власних класів та використання об'єктів, що належать цим класам. У пам'яті залишилися основні поняття: для оголошення класу використовується ключове слово «class», а за замовчуванням класи мають внутрішній доступ, що означає, що вони доступні лише у межах поточного проекту. Якщо потрібно зробити клас загальнодоступним для інших проектів, можна використати ключове слово «public». Це саме ключове слово застосовується до класів форм, які використовуються в розробці Windows-додатків.

Крім модифікаторів доступу, існують ще два ключові слова для опису класів: «abstract» (абстрактний) та «sealed» (ізольований). Вони використовуються взаємовиключною. Абстрактні класи не можуть бути інстанційовані, але можуть мати абстрактні методи (зі словом «abstract»), які визначають прототипи, але не мають реалізації. Ці методи повинні бути

перевизначені в похідних класах (зі словом «override»). З іншого боку, ізольовані класи не можуть бути успадкованими.

Отже, освоєно основи роботи з класами, модифікаторами доступу та деякими ключовими словами, які дозволяють вам гнучко визначати структуру та поведінку класів у програмному проекті.

Ієрархія в програмах Windows є важливим аспектом об'єктозорієнтованого програмування. Вона використовується для організації класів та їх взаємозв'язків з метою створення структурованих і модульних програм.

У програмах Windows ієрархія часто відображається у вигляді деревоподібної структури, де кожен вузол представляє клас або об'єкт, а зв'язки між вузлами вказують на спадковість (успадкування) або композицію.

На вершині ієрархії знаходиться базовий клас, який визначає загальні властивості і методи для певної групи класів. Цей клас може мати похідники, які успадковують його властивості та методи, а також додають власні специфічні реалізації.

У програмах Windows часто використовуються ієрархії для розробки віконних додатків. Наприклад, базовий клас може представляти загальну форму (Form), а похідні класи будуть відповідати конкретним формам додатків. Така ієрархія дозволяє використовувати спільні функціональні можливості базового класу і одночасно розширювати їх у специфічних формах.

Додатково, ієрархія може включати інші класи, які представляють різні елементи інтерфейсу користувача, такі як кнопки, текстові поля, списки тощо. Ці класи можуть успадковувати властивості та методи від базових класів, забезпечуючи єдинобразний і зручний спосіб управління елементами інтерфейсу.

Ієрархія в програмах Windows дозволяє створювати багатофункціональні та розширювані програми, де класи спадкоємці можуть додавати власні

функції та розширювати базовий функціонал. Це сприяє структурованості, повторному використанню коду і полегшує розвиток програмного продукту.

Окрім того, ієрархія в програмах Windows також використовується для організації компонентів і бібліотек. Компоненти можуть бути створені у вигляді класів і включати в себе різноманітні функціональні можливості, які можуть бути використані у різних додатках. Це дозволяє розподілити функції програмного продукту на логічно зв'язані компоненти, спрощує підтримку і розвиток системи.

Ієрархія дозволяє організувати код у логічні групи, розширювати та наслідувати функціонал, спрощує управління та розвиток програмного продукту. Навички розуміння ієрархії класів та їх використання є важливими для розробки ефективних та масштабованих програм під платформу Windows.

3.2 Використання поліморфізму у формі застосунку

Поліморфізм є одним з ключових принципів об'єктно-орієнтованого програмування і відіграє важливу роль у розробці програмних застосунків. Використання поліморфізму дозволяє зробити програму більш гнучкою та розширюваною, забезпечуючи можливість обробки різних типів об'єктів за допомогою спільного інтерфейсу.

У контексті форми застосунку, поліморфізм може бути використаний для обробки різних подій або взаємодії з елементами інтерфейсу. Наприклад, в програмі створення програмного інструментарію викладача розділу «Статика», можуть бути різні типи об'єктів, такі як кнопки, поле введення тексту, малювальна область тощо.

За допомогою поліморфізму, можна створити загальний інтерфейс або базовий клас для цих об'єктів, що визначає спільні методи чи властивості. Наприклад, може бути базовий клас «Елемент інтерфейсу» з методами, такими як «Натиснути», «Редагувати», «Зобразити» та іншими.

Кожен конкретний тип об'єкту, такий як кнопка чи поле введення тексту, може успадкувати цей базовий клас і реалізувати власні методи специфічні для свого типу. Таким чином, коли користувач взаємодіє з елементами інтерфейсу, програма може викликати спільні методи через загальний інтерфейс, що спрощує обробку подій та взаємодію з різними об'єктами.

Поліморфізм також дозволяє легко розширювати функціонал програми. Наприклад, якщо у майбутньому було б потрібно додати новий тип елементу інтерфейсу, такий як слайдер або список, то достатньо створити новий клас, що успадковує базовий клас «Елемент інтерфейсу» і реалізовує власні методи для цього типу. Всі існуючи частини коду, які використовують поліморфізм, будуть продовжувати працювати без змін, оскільки вони взаємодіють з новим елементом інтерфейсу через загальний інтерфейс.

Поліморфізм також сприяє зменшенню пов'язаності між класами, що покращує модульність і розширюваність програми. Кожен тип об'єкту може мати свою власну реалізацію методів, не впливаючи на решту системи.

Таким чином, використання поліморфізму в формі застосунку дозволяє створити гнучку, розширювану програму, де різні типи елементів інтерфейсу можуть взаємодіяти з використанням спільного інтерфейсу або базового класу. Це спрощує розробку і підтримку програми, а також забезпечує зручну взаємодію з користувачем та обробку різноманітних подій у програмі.

3.3 Використання об'єктозорієнтованого програмування

Робота [8] свідчить про те, що об'єктозорієнтоване програмування - це парадигма програмування, яка дозволяє створювати програми на основі концепцій об'єктів, які взаємодіють один з одним. Воно використовує об'єкти, які є екземплярами класів, і дозволяє використовувати спадкування, поліморфізм, інкапсуляцію та інші принципи для організації коду і розв'язання проблем.

Основні принципи Об'єктозорієнтованого програмування включають:

1. Спадкування: Можливість створювати нові класи на основі існуючих, успадковуючи їхні властивості та методи. Спадкування дозволяє створювати ієрархію класів, де класи-нащадки успадковують функціональність від батьківських класів.

2. Поліморфізм: Можливість використовувати об'єкти підкласів через їхні базові класи. Це дозволяє одному об'єкту виконувати різні дії залежно від його типу.

3. Інкапсуляція: Принцип, за яким дані і функції, які з ними пов'язані, об'єднані в один об'єкт і захищені від прямого доступу. Інкапсуляція дозволяє приховати внутрішні деталі об'єкта і забезпечити доступ до них тільки через визначені методи.

4. Абстракція: Процес виділення основних характеристик об'єкта і визначення інтерфейсу для взаємодії з ним. Абстракція дозволяє спростити складність системи, фокусуючись на суттєвих аспектах.

Об'єктозорієнтоване програмування сприяє підвищенню модульності, повторного використання коду, легкості розширення і підтримки, а також полегшує співпрацю великих команд розробників. Він дозволяє створювати більш структуровані та зрозумілі програми, що полегшує їх розуміння та підтримку в подальшому.

Також воно широко використовується в багатьох мовах програмування, включаючи C++, Java, C#, Python і багато інших. Використання ООП дозволяє розробникам ефективно створювати складні системи, які можуть бути легко модифіковані та розширені у майбутньому.

3.4 Реалізація класів

У роботі [9] вказано , що в програмуванні мовою C# класи є основною будівельною одиницею програмного коду. Класи дозволяють організувати дані та функціональність в одну структуру, що спрощує розробку та

підтримку програм. Класи в C# визначаються за допомогою ключового слова "class" та можуть мати різні члени, такі як поля (змінні), методи (функції) та властивості. Поля представляють дані, які зберігаються в класі, методи виконують дії над цими даними, а властивості надають доступ до цих даних ззовні класу.

Аналізуємо, як приклад, код класу, див. додаток А, Figure_1_1 схеми 1 а , аналогічно з яким створені всі інші.

Даний код є прикладом визначення різних фігур та їх відображення за допомогою бібліотеки System.Drawing у середовищі Windows Forms.

Основна частина коду складається з оголошень класів, які відповідають окремим фігурам, абстрактного класу Figure_1_1 та його похідних класів. В класі Figure_1_1 оголошуються основні властивості фігури (колір, координати, назва) та абстрактний метод Draw, який буде реалізований у похідних класах для відображення конкретної фігури.

Класи Element_Arc_1_1, Element_Line_1_1, Element_Line_Arrow_1_1, Element_Lines_Arrows_1_1, Element_Lines_Incline_1_1, Element_Text_1_1 та Element_Lines_Incline_v_1_1 успадковують клас Figure_1_1 та реалізовують метод Draw залежно від типу фігури.

1. Element_Arc_1_1:

Цей клас представляє дугу на площині. Має атрибути center (центр дуги), radius (радіус дуги), start_angle (початковий кут) і end_angle (кінцевий кут). Метод length() обчислює довжину дуги, використовуючи формулу 2 * pi * radius * (end_angle - start_angle) / 360. Метод area() обчислює площу дуги, використовуючи формулу pi * radius^2 * (end_angle - start_angle) / 360.

2. Element_Line_1_1:

Цей клас представляє пряму лінію на площині. Має атрибути point1 і point2, які визначають початок і кінець лінії. Метод length() обчислює довжину лінії, використовуючи формулу відстані між двома точками у просторі. Метод slope() обчислює нахил лінії, використовуючи формулу (y2 -

y1) / (x2 - x1). Метод midpoint() знаходить середину лінії, обчислюючи середнє арифметичне координат по осі X та Y для точок point1 і point2.

3. Element_Line_Arrow_1_1:

Цей клас представляє пряму лінію зі стрілками на кінцях на площині. Наслідується від класу Element_Line_1_1. Має додатковий атрибут arrow_length (довжина стрілки). Метод draw() візуалізує лінію зі стрілками.

4. Element_Lines_Arrows_1_1:

Цей клас представляє ряд ліній зі стрілками на площині. Має атрибути start_point (початкова точка) і end_point (кінцева точка), які визначають початок і кінець ряду ліній. Має додатковий атрибут arrow_size (розмір стрілки). Метод length() обчислює довжину ряду ліній, використовуючи формулу відстані між початковою і кінцевою точками. Метод draw() візуалізує ряд ліній зі стрілками на площині, використовуючи початкову і кінцеву точки та розмір стрілки.

5. Element_Lines_Incline_1_1:

Цей клас представляє нахилений ряд ліній на площині. Має атрибути start_point (початкова точка) і end_point (кінцева точка), які визначають початок і кінець ряду ліній. Метод length() обчислює довжину ряду ліній, використовуючи формулу відстані між початковою і кінцевою точками. Метод slope() обчислює нахил ряду ліній, використовуючи формулу (y2 - y1) / (x2 - x1). Метод midpoint() знаходить середину ряду ліній, обчислюючи середнє арифметичне координат по осі X та Y для початкової і кінцевої точок.

6. Element_Text_1_1:

Цей клас представляє текстовий елемент на площині. Має атрибути position (положення тексту) і text (текстовий рядок). Метод draw() візуалізує текстовий елемент на площині.

7. Element_Lines_Incline_v_1_1:

Цей клас представляє вертикальний нахилений ряд ліній на площині. Наслідується від класу Element_Lines_Incline_1_1. Має додатковий атрибут

line_count (кількість ліній у ряді). Метод draw() візуалізує вертикальний нахилений ряд ліній на площині.

Клас Figure_1_1 є абстрактним класом, який містить загальні властивості і методи для всіх фігур. У ньому оголошено захищені поля для зберігання інформації про кольори, координати і назву фігури.

У цьому фрагменті коду лістингу 6.1 описано метод DrawEllipse, який малює еліпс на графічному контексті g. Він приймає параметри myPen, який визначає властивості олівця для малювання, і координати та розміри еліпса (x - 4, y - 4, 8, 8).

```
 g.DrawEllipse(myPen, x - 4, y - 4, 8, 8);
        }
        private int a;
        private int b;
    }
}
```

Лістинг 3.1 – код DrawEllipse

Також в цьому фрагменті оголошено дві приватні змінні a і b, які можуть використовуватися в інших частинах коду, але самі не мають жодної логіки чи функціоналу в цьому фрагменті.Крім того, в цьому класі є конструктори, які ініціалізують ці поля і методи для отримання/зміни значень цих полів. Також в цьому класі визначений абстрактний метод Draw, який потрібно реалізувати у всіх похідних класах.

Element_Text_1_1: Цей клас представляє текстовий елемент на площині. Має атрибути position (положення тексту) і text (текстове значення), які визначають місцезнаходження і вміст тексту. Має метод draw() для візуалізації текстового елемента на площині.

Element_Lines_Incline_v_1_1: Цей клас представляє наклонену пару ліній на площині. Має атрибути start_point (початкова точка), length (довжина) і angle (кут нахилу), які визначають геометрію наклонених ліній. Має метод

draw() для візуалізації наклоненої пари ліній на площині, використовуючи геометрію ліній.

Ці класи додають різноманітні типи елементів до бібліотеки для малювання на площині. Кожен клас має свої атрибути і методи, які дозволяють визначати їх геометрію та візуалізувати їх на площині. Можна використовувати ці класи для створення складніших малюнків і додавати їх графічної бібліотеки у майбутньому.

3.5 Додання формул до програми

Код з лістингу 3.2 представляє функцію з назвою «formula_a_1», яка виконує обчислення за певною формулою. Основна мета цієї формули полягає в обчисленні різних значень (X_a, Y_a, M_a, R_a) на основі заданих змінних (P, q, M) та встановлення цих значень в текстові поля (X_A, Y_A, M_A, R_A) на інтерфейсі програми.

```
   void formula_a_1()        {
               var X_a = P * Utils.cosine(60);            var Y_a
= P * Utils.sinine(60)+ 2*q;
               var M_a = M + P + 4 * q;            var R_a =
Math.Sqrt(Math.Pow(X_a, 2.0) + Math.Pow(Y_a, 2.0));
               X_A.Text = X_a.ToString("0.00");
Y_A.Text = Y_a.ToString("0.00");
               M_A.Text = M_a.ToString("0.00");
R_A.Text = R_a.ToString("0.00");
               X_B.Text = "0.00";            Y_B.Text = "0.00";
               M_B.Text = "0.00";            R_B.Text = "0.00";
           }
```

Лістинг 3.2 – Код до формули А

Основні кроки цього коду:

1. Обчислення значення X_a: використовується змінна P, помножена на косинус 60 градусів (виклик функції Utils.cosine(60)). Результат зберігається у змінній X_a.

2. Обчислення значення Y_a: використовується змінна P, помножена на синус 60 градусів (виклик функції Utils.sinine(60)), додана до значення 2*q. Результат зберігається у змінній Y_a.

3. Обчислення значення M_a: використовуються змінні M, P та 4*q. Значення M_a обчислюється шляхом додавання цих змінних.

4. Обчислення значення R_a: використовується формула обчислення квадратного кореня з суми квадратів X_a та Y_a (використовуються функції Math.Pow() та Math.Sqrt()). Результат зберігається у змінній R_a.

5. Значення X_a, Y_a, M_a та R_a форматуються у рядок з точністю до двох знаків після коми за допомогою методу ToString(«0.00»).

6. Значення X_a, Y_a, M_a, R_a присвоюються відповідним текстовим полям X_A, Y_A, M_A, R_A на інтерфейсі програми.

Значення текстових полів X_B, Y_B, M_B, R_B встановлюються як «0.00».

Код з лістингу 3.3 представляє реалізацію функції «formula_b_1», яка виконує обчислення для визначення значень реакцій опор твердого тіла.

```
   void formula_b_1()        {
               var X_a = P * Utils.cosine(60);            var X_b
= 0;
                          var Y_b = 0.25*(M+P+4*q);
               var Y_a = 0.5*P*Math.Sqrt(3)+2*q-Y_b;
               var M_a = 0;            var M_b = 0;
               var R_a = Math.Sqrt(Math.Pow(X_a, 2.0) +
Math.Pow(Y_a, 2.0));
               var R_b = Y_b;            M_A.Text =
M_a.ToString("0.00");
               X_A.Text = X_a.ToString("0.00");
Y_A.Text = Y_a.ToString("0.00");
```

```
                R_A.Text = R_a.ToString("0.00");
M_B.Text = M_b.ToString("0.00");
                X_B.Text = X_b.ToString("0.00");
Y_B.Text = Y_b.ToString("0.00");
                M_B.Text = M_b.ToString("0.00");
R_B.Text = R_b.ToString("0.00");

            }
```

Лістинг 3.3 – Код до формули Б

Вхідні дані для розрахунків включають змінні P, M і q.

Зокрема, дана формула розраховує значення координат X і Y для точок A і B, а також величину моменту M_a, M_b та радіус R_a, R_b для відповідних точок.

1. Змінні X_a і X_b визначаються на основі формул P * Utils.cosine(60) та 0 відповідно. Вони зображають координати точок A і B по осі X.

2. Змінна Y_b обчислюється за допомогою формули 0.25 * (M + P + 4*q) і відповідає координаті точки B по осі Y.

3. Змінна Y_a обчислюється за формулою 0.5 * P * Math.Sqrt(3) + 2*q - Y_b і представляє координату точки A по осі Y.

4. Змінні M_a і M_b ініціалізуються нульовими значеннями, оскільки в даному контексті немає моментів у точках A і B.

5. Змінні R_a і R_b визначаються відповідно за допомогою формул Math.Sqrt(Math.Pow(X_a, 2.0) + Math.Pow(Y_a, 2.0)) і Y_b. Вони зображають величину радіуса вектора для точок A і B.

6. Значення змінних M_a, X_a, Y_a, R_a присвоюються відповідним текстовим полям M_A, X_A, Y_A, R_A, які, ймовірно, представляють відповідні візуальні елементи на інтерфейсі програми.

7. Аналогічно, значення змінних M_b, X_b, Y_b, R_b присвоюються відповідним текстовим полям M_B, X_B, Y_B, R_B.

Цей код також демонструє використання формул для розрахунку реакцій опор твердого тіла та зображення результатів ба вставляючи їх у відповідь у текстові поля на інтерфейсі програми.

У лістингу 3.4 розглянуто функцію «formula_v_1», яка містить реалізацію формули для обчислення значень реакцій опор твердого тіла. Давайте розглянемо детальніше цю формулу:

```
   void formula_v_1()        {
                            var R_b = (P+4*q)/(2*Math.Sqrt(3)-1);
                var X_a = 0.5*R_b+0.5*P;
                var X_b = 0;
                var Y_b = 0;            var Y_a = 0.5 *
Math.Sqrt(3) * P + 2 * q - 0.5 * Math.Sqrt(3) * R_b;
                var M_a = 0;
                var M_b = 0;            var R_a =
Math.Sqrt(Math.Pow(X_a, 2.0) + Math.Pow(Y_a, 2.0));
                M_A.Text = M_a.ToString("0.00");
                X_A.Text = X_a.ToString("0.00");
Y_A.Text = Y_a.ToString("0.00");
                R_A.Text = R_a.ToString("0.00");
M_B.Text = M_b.ToString("0.00");
                X_B.Text = X_b.ToString("0.00");
Y_B.Text = Y_b.ToString("0.00");
                M_B.Text = M_b.ToString("0.00");
R_B.Text = R_b.ToString("0.00");
           }
```

Лістинг 6.4 – Код до формули B

1. Змінна R_b обчислюється за допомогою формули (P + 4*q) / (2 * Math.Sqrt(3) - 1). Ця формула виконує розрахунок радіуса R_b відповідно до вхідних параметрів P і q.

2. Змінна X_a обчислюється за формулою 0.5 * R_b + 0.5 * P. Вона відображає координату точки A по осі X.

3. Змінна X_b ініціалізується значенням 0, що вказує на те, що точка B знаходиться на початку координат.

4. Змінна Y_b ініціалізується значенням 0, оскільки в даному контексті немає зміщення по осі Y для точки B.

5. Змінна Y_a обчислюється за формулою 0.5 * Math.Sqrt(3) * P + 2 * q - 0.5 * Math.Sqrt(3) * R_b. Вона представляє координату точки A по осі Y.

6. Змінні M_a і M_b ініціалізуються нульовими значеннями, оскільки в даному контексті немає моментів у точках A і B.

7. Змінна R_a обчислюється за допомогою формули Math.Sqrt(Math.Pow(X_a, 2.0) + Math.Pow(Y_a, 2.0)). Вона відображає величину радіуса вектора для точки A.

8. Значення змінних M_a, X_a, Y_a, R_a присвоюються відповідним текстовим полям M_A, X_A, Y_A, R_A, які, імовірно, представляють відповідні візуальні елементи на інтерфейсі програми.

9. Значення змінних M_b, X_b, Y_b, R_b присвоюються відповідним текстовим полям M_B, X_B, Y_B, R_B, які, імовірно, представляють відповідні візуальні елементи на інтерфейсі

ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

Ця інструкція призначена для надання детальних вказівок та кроків користувачам щодо використання програмного інструментарію викладача розділу «Статика». Вона створена з метою допомогти вам у виконанні певних завдань та розв'язанні статичних ситуацій шляхом використання програми.

У цій інструкції будуть наведені основні кроки та функціональні можливості програмного інструментарію, які дозволять вирішувати статичні задачі, використовуючи передбачені формули та методи.

Ця інструкція буде включати крок за кроком опис процесу використання програми, включаючи розміщення елементів на інтерфейсі, введення значень та отримання результатів. Також буде надано пояснення щодо використання різних формул та їх зв'язку з розрахунками реакцій опор твердого тіла.

Після відкриття у програмі нас зустрічає екран із першою схемою та її стартовими значеннями (рис. 4.1).

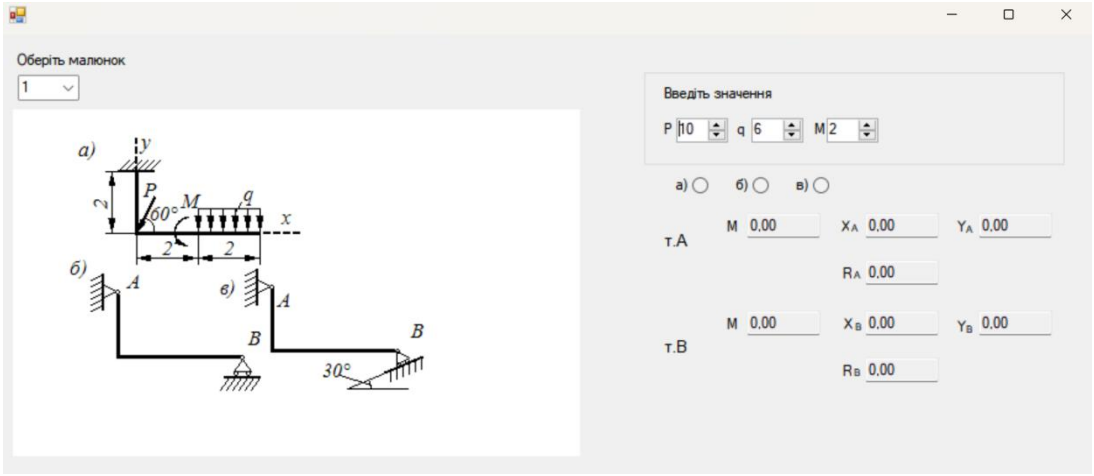

Рисунок 4.1 – Вигляд програми після запуску

У верхньому лівому куті одразу можна обрати малюнок (схему) за варіантом (рис. 4.2), який відображає номер статичної ситуації, що буде розглядатись.

Обрання малюнка в початковому етапі дозволяє вам зосередитись на конкретній ситуації, яку ви бажаєте проаналізувати, і допомагає вам

зорієнтуватись у подальших кроках використання програмного інструментарію.

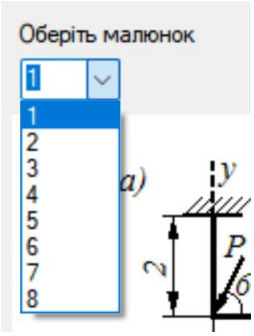

Рисунок 4.2 – Перелік усіх номерів варіантів малюнків (схем)

Після обраного варіанту відповідна схема буде зображена на білому прямокутнику, який служить робочою областю для подальшого аналізу. У правій частині екрану ви знайдете контейнер з вхідними параметрами задачі: P (сила), M (момент) та q (розподіл навантаження). Ви маєте можливість ввести відповідні значення для цих параметрів, які впливають на статичну ситуацію.

Після введення вхідних параметрів вам потрібно обрати відповідний варіант (А, Б або В), що відображає конкретну задачу або умови розв'язання. Вибір варіанту дозволяє програмі знати, які конкретні відповіді ви бажаєте отримати для даної статичної ситуації (рис. 4.3).

Після здійснення вибору варіанту програма виведе відповіді на екран. За замовчуванням, всі відповіді будуть дорівнювати нулю, що вказує на відсутність силових реакцій та переміщень. Однак, залежно від введених параметрів та обраного варіанту, програма зможе вивести відповіді, що відображають силові реакції, моменти, переміщення та інші характеристики статичної ситуації.

Загалом, ця інтерактивна програма надає вам зручний інструмент для аналізу та розв'язання статичних задач, де ви можете ввести вхідні параметри,

обрати варіант та отримати відповіді, що відображають стан тіла в заданій статичній ситуації.

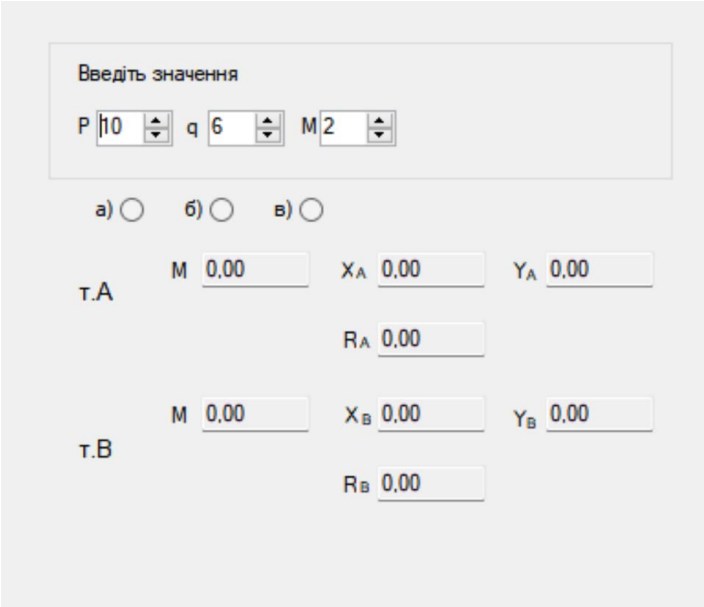

Рисунок 4.3 – Вхідні та вихідні данні

Вхідні значення можна змінювати завдяки NumericUpDown, щоб змінювати по одному символу за раз і користувач не міг ввести недопустимі значення.

Після задання потрібних значень та обравши потрібний варіант (А, Б або В) то програма вираховує відповіді, а також домальовує сили реакцій червоним кольором на відповідній схемі (рис. 4.4):

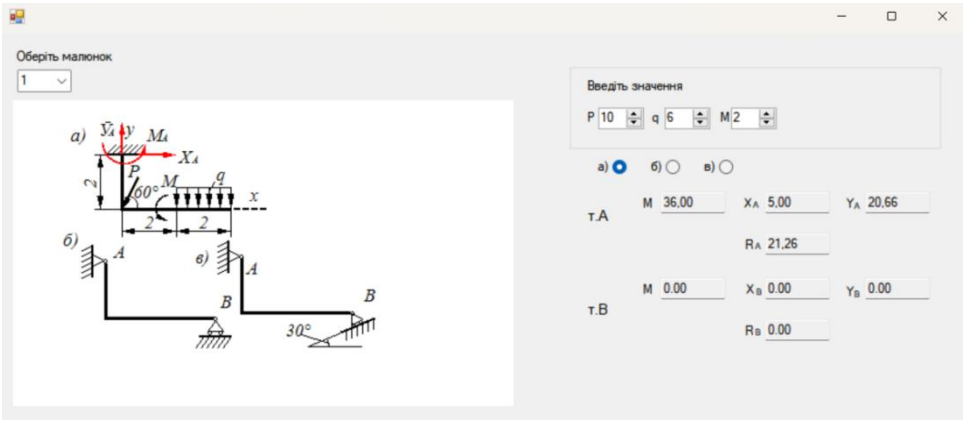

Рисунок 4.4 – Відповіді та сили реакцій

Вхідні значення можна змінювати за допомогою елемента керування NumericUpDown, що дозволяє змінювати значення по одному символу за раз. Це дозволяє користувачеві точно налаштовувати параметри задачі і уникнути введення недопустимих значень.

Після введення потрібних значень і вибору відповідного варіанту (А, Б або В), програма автоматично виконує розрахунки для отримання відповідей. Крім того, відповідна схема, що відображає статичну ситуацію, буде додатково домальована. Наприклад, сили реакції на опори будуть показані червоним кольором на відповідних місцях схеми (рис. 4.4). Це дозволяє вам візуально сприйняти результати розрахунків та зрозуміти, які сили діють на тіло в даній статичній ситуації.

Такий підхід з використанням NumericUpDown та додаткового підсвічування сил реакцій на схемі сприяє зручності та точності вводу параметрів, а також дозволяє отримати ясне візуальне представлення результатів розрахунків. Подальший аналіз та сприйняття статичної ситуації стає більш зрозумілим та ефективним для користувача.

ВИСНОВКИ

Завдяки розробленій програмі для визначення реакцій опор твердого тіла, можна отримати потужний інструмент, який може допомогти у багатьох аспектах інженерної роботи.

Швидкість та точність: Програма дозволяє швидко та точно визначити реакції опор, що займає значну кількість часу та зусиль, якщо робити це вручну. Вона автоматизує процес обчислення та забезпечує високу точність результатів.

Ефективність проектування: З використанням програмного інструментарію можна швидше створювати та аналізувати різні конструкції. Щоб проводити широкий спектр розрахунків та експериментувати з різними параметрами, щоб знайти оптимальні рішення тепер потрібно набагато менше часу.

Мінімізація помилок: Ручні розрахунки можуть призвести до помилок, особливо при роботі з складними формулами. Програма забезпечує автоматичні обчислення на основі введених формул, що зменшує ймовірність помилок та забезпечує надійні результати.

Візуалізація результатів: Програма дозволяє візуально представляти вигляд схем та силу реакцій тих схем, за для наглядного розуміння результатів розрахунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Яблонський А.А., Курс теоретичної механіки Ч.1 / Яблонський А.А, Нікіфорова В.М. // Москва, 1966 – 174 с.

2. Павловський М.А., Теоретична Механіка // Київ: «ТЕХНІКА», 2002 – 88 с.

3. Апостолюк О. С., Теоретична механіка: Збірник задач / О. С. Апостолюк, В. М. Воробйов, Д. І. Ільчишина та ін.// Київ: «Техніка», 2007 – 144 с.

4. Березін Л.М., Теоретична механіка / Березін Л.М., Кошель С.О.// Київ: «Центр учбової літератури», 2020 – 64 с.

5. Яскілка М.Б., Збірник завдань для розрахунково-графічних робіт з теоретичної механіки/ Київ: Вища школа: «Веселка», 1999. – 76 с.

6. Дібрівний О.А., Вступ до об’єктно-орієнтованого програмування C#/ Дібрівний О.А., Гребенюк В.В.// Київ: Державний університет телекомунікацій, 2018. – 190 с.

7. Коноваленко І.В., Програмування мовою C# 6.0// Тернопіль: ТНТУ, 2016. – 227 с.

8. Бублик В.В. Об'єктно-орієнтоване програмування// Київ: Підручник: «ІТ-книга», 2015 – 21 с.

9. Кравець П.О., Об'єктно-орієнтоване програмування// Львів: Видавництво Львівської політехніки, 2012 – 53 с.

ДОДАТОК А

Код Figure_1_1 для створення схеми 1 а

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsFormsApp1
{
    abstract class Figure_1_1 //клас 1 фігури
    {
        protected Color color; // зберігаємо інформацію про компоненти фігури
        protected string name;
        protected int x;
        protected int y;
        public Figure_1_1()
        {
            color = Color.Black;
            x = 0;
            y = 10;
            name = "";
        }
        public Figure_1_1(int k1, int k2, string n, Color c, int s,bool p)
        {

            color = c;
            x = k1;
            y = k2;
            name = n;
        }
        // функції отримання та занесення інформації про фігури
        public void SetColor(Color c) { color = c; }
        public Color GetCh() { return color; }
        public void SetX(int k) { x = k; }
        public int GetX() { return x; }
        public void SetY(int s) { y = s; }
        public int GetY() { return y; }
        public void SetName(string n) { name = n; }
        public string GetName() { return name; }
        abstract public void Draw(Graphics g);

}
    class Element_Arc_1_1 : Figure_1_1 // дуга
    {
        public Element_Arc_1_1()
        {
            a = 0; b = 0;
        }
        public Element_Arc_1_1(int d1, int d2)
        {
            a = d1;
            b = d2;
        }

        public Element_Arc_1_1(int k1, int k2, int d1, int d2, string n, Color c, int s, bool
p, int l1, int l2) : base(k1, k2, n, c, s, p)
        {
            a = d1;
            b = d2;
            l = l1;
            m = l2;
            arrow = p;


        }
        public void SetA(int d)
        {
            a = d;
```

Лістинг А.1, лист 1

```
            }

            public int GetA()
            {
                return a;
            }



            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }

            override public void Draw(Graphics g)
            {

                Pen myPen = new Pen(color, 1);
                if (arrow)
                {
                    myPen.Width = 1.5f;
                    myPen.CustomStartCap = new AdjustableArrowCap(2.5f, 6.0f); // кінець стрілка
                }
                else
                {

                }

                Rectangle rect = new Rectangle(x, y, a, b); //координати дуги
                g.DrawArc(myPen, rect, l, m); // малюємо дугу

            }
            private int a;
            private int b;
            private int l;
            private int m;
            private bool arrow;


        }
        class Element_Line_1_1 : Figure_1_1 // наслідуємо класс фігури 1
        {
            public Element_Line_1_1() // лінія
            {
                a = 0;
                b = 0;

    }
    public Element_Line_1_1(int d1, int d2)
    {
    a = d1;
    b = d2;
    }
    public Element_Line_1_1(int k1, int k2, int d1, int d2, string n, Color c, int s, bool p) :
base(k1, k2, n, c, s, p) //
    {
    a = d1;
    b = d2;
    size = s;
    dotted_line = p;
    }
    public void SetA(int d)
    {
    a = d;
    }
    public int GetA()
    {
    return a;
    }
```

Лістинг А.1, лист 2

```
    public void SetB(int d)
    {
    b = d;
    }
    public int GetB()
    {
                return b;
            }
            override public void Draw(Graphics g)
            {
                Pen myPen = new Pen(color, 1);
                myPen.Width = size;
                if (!dotted_line) // якщо ні, звичайна лінія
                {
                    myPen.DashStyle = DashStyle.Solid;
                }
                else//якщо та, пунктирна лінія
                {
                    myPen.DashStyle = DashStyle.Dash;
                }


                g.DrawLine(myPen, x, y, a, b); // малюємо лінію по кординатах

            }
            private int a;
            private int b;
            private int size;
            private bool dotted_line;
        }

        class Element_Line_Arrow_1_1 : Figure_1_1
        {
            public Element_Line_Arrow_1_1() //лінія зі стрілкою
            {
                a = 0; b = 0;
            }
            public Element_Line_Arrow_1_1(int d1, int d2)
            {
                a = d1;
                b = d2;
            }
            public Element_Line_Arrow_1_1(int k1, int k2, int d1, int d2, string n, Color c, int
s, bool p) : base(k1, k2, n, c, s, p)
            {
                a = d1;
                b = d2;
                arrow = p;
                size = s;

            }
            public void SetA(int d)
            {
                a = d;
            }
            public int GetA()
            {
                return a;
            }



            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }
            override public void Draw(Graphics g)
            {
                Pen myPen = new Pen(color, 1);

                if (!arrow)
```

Лістинг А.1, лист 3

```
                {
                    myPen.Width = 1;
                    myPen.CustomEndCap = new AdjustableArrowCap(3, 6);
                    myPen.CustomStartCap = new AdjustableArrowCap(3, 6);
                }
                else
                {
                    myPen.Width = 2;
                    myPen.CustomStartCap = new AdjustableArrowCap(3, 6);
                }
                g.DrawLine(myPen, x, y, a, b);
            }
            private int a;
            private int b;
            private bool arrow;
            private int size;

        }

        class Element_Lines_Arrows_1_1 : Figure_1_1
        {
            public Element_Lines_Arrows_1_1() //лінія зі стрілками
                a = 0; b = 0;
            }
            public Element_Lines_Arrows_1_1(int d1, int d2)
            {
                a = d1;
                b = d2;
            }
            public Element_Lines_Arrows_1_1(int k1, int k2, int d1, int d2, string n, Color c,
int s, bool p) : base(k1, k2, n, c, s, p)
            {
                a = d1;
                b = d2;

            }
            public void SetA(int d)
            {
                a = d;
            }
            public int GetA()
            {
                return a;
            }



            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }

            override public void Draw(Graphics g)
            {
                Pen myPen = new Pen(color, 1);
                myPen.Width = 1; // розмір лінії
                int xx = x;
                int aa = a;
                g.DrawLine(myPen, x, y, a + 50, b - 20);

                myPen.CustomEndCap = new AdjustableArrowCap(3, 7);

                for (int i = 0; i < 6; i++)
                {
                    g.DrawLine(myPen, xx, y, aa, b);
                    xx += 10;
                    aa += 10;
                }
```

Лістинг А.1, лист 4

```
            }
            private int a;
            private int b;

        }
        class Element_Lines_Incline_1_1 : Figure_1_1
        {
            public Element_Lines_Incline_1_1() // за багатьма лініями
            {
                a = 0; b = 0;
            }
            public Element_Lines_Incline_1_1(int d1, int d2)
            {
                a = d1;
                b = d2;
            }
            public Element_Lines_Incline_1_1(int k1, int k2, int d1, int d2, string n, Color c,
int s, bool p) : base(k1, k2, n, c, s, p)
            {
                a = d1;
                b = d2;

            }
            public void SetA(int d)
            {
                a = d;
            }
            public int GetA()
            {
                return a;
            }



            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }
            override public void Draw(Graphics g)
            {
                Pen myPen = new Pen(color, 1);
                myPen.Width = 1;
                int xx = x;
                int aa = a;

                for (int i = 0; i < 6; i++) // малюємо 6 ліній
                {
                    g.DrawLine(myPen, xx, y, aa + 9, b - 9);
                    xx += 5;
                    aa += 5;
                }
                myPen.Width = 2;
                g.DrawLine(myPen, x + 30, y, a, b);
            }
            private int a;
            private int b;

        }
        class Element_Text_1_1 : Figure_1_1
        {
            public Element_Text_1_1() //текст
            {
                a = 0; b = 0;
            }
            public Element_Text_1_1(int d1, int d2)
            {
                a = d1;
                b = d2;
            }
```

Лістинг А.1, лист 5

```
            public Element_Text_1_1(int k1, int k2, int d1, int d2, string n, Color c, int s,
bool p) : base(k1, k2, n, c, s, p)
            {
                a = d1;
                b = d2;
                t = n;
                r = p;
                size = s;
            }
            public void SetA(int d)
            {
                a = d;
            }
            public int GetA()
            {
                return a;
            }

            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }
            override public void Draw(Graphics g)
            {


                String drawString = t;

                // Create font and brush.
                Font drawFont = new Font("Times New Roman", size, FontStyle.Italic); // шрифт та
розмір

                SolidBrush drawBrush = new SolidBrush(color);

                // Create point for upper-left corner of drawing.
                g.ResetTransform();

                // Set format of string.
                StringFormat drawFormat = new StringFormat();

                if (r)
                {
                    g.RotateTransform(-90); // перевернутий текст

                    g.DrawString(drawString, drawFont, drawBrush, x - 210, y - 10, drawFormat);
                }
                Else
                {
                    g.ResetTransform();
                    g.DrawString(drawString, drawFont, drawBrush, x, y, drawFormat);
                }
            }
            private int a;
            private int b;
            private string t;
            private bool r;
            private int size;

        }
        class Element_Lines_Incline_v_1_1 : Figure_1_1
        {
            public Element_Lines_Incline_v_1_1() // вертикальний багатоліний компонент
            {
                a = 0; b = 0;
            }
            public Element_Lines_Incline_v_1_1(int d1, int d2)
            {
                a = d1;
                b = d2;
```

Лістинг А.1, лист 6

```
            }
            public Element_Lines_Incline_v_1_1(int k1, int k2, int d1, int d2, string n, Color c,
int s, bool p) : base(k1, k2, n, c, s, p)
            {
                a = d1;
                b = d2;
            }
            public void SetA(int d)
            {
                a = d;
            }
            public int GetA()
            {
                return a;
            }



            public void SetB(int d)
            {
                b = d;
            }
            public int GetB()
            {
                return b;
            }

            override public void Draw(Graphics g)
            {
                Pen myPen = new Pen(color, 1);
                myPen.Width = 2;


                g.DrawLine(myPen, x, y, a, b);//85, 250, 85, 220
                myPen.Width = 1;
                g.DrawLine(myPen, x, y - 30, a - 10, b + 5);
                g.DrawLine(myPen, x, y - 25, a - 10, b + 10);
                g.DrawLine(myPen, x, y - 20, a - 10, b + 15);
                g.DrawLine(myPen, x, y - 15, a - 10, b + 20);
                g.DrawLine(myPen, x, y - 10, a - 10, b + 25);
                g.DrawLine(myPen, x, y - 5, a - 10, b + 30);
                var points = new Point[]
               {
               new Point(x+10, y-16),
               new Point(x, y-11),
```

Лістинг А.1, лист 16

```
               new Point(x, y-21)
                    };

                g.DrawPolygon(myPen, points);
                g.DrawEllipse(myPen, x + 10, y - 17, 3, 3);

            }
            private int a;
            private int b;

        }
    }
```

Лістинг А.1, лист 7