1. Властивості основної пентади програмування: Основна пентада програмування включає п'ять ключових властивостей, що визначають сутність процесу розробки програмного забезпечення:

    - Точність (Correctness): Програма повинна виконувати поставлені завдання відповідно до специфікацій і вимог. Це означає, що для заданих вхідних даних програма має видавати правильні вихідні дані.
    - Ефективність (Efficiency): Програма повинна використовувати обчислювальні ресурси (час, пам'ять) оптимальним чином. Ефективність може вимірюватися часовою складністю та просторовою складністю алгоритму.
    - Надійність (Reliability): Програма повинна стабільно працювати в різних умовах, включаючи неочікувані вхідні дані або помилки користувача. Надійна програма має бути стійкою до збоїв і передбачуваною у своїй поведінці.
    - Зручність використання (Usability): Програма повинна бути легкою для розуміння та використання кінцевими користувачами. Інтуїтивно зрозумілий інтерфейс, чітка документація та легкість навігації є важливими аспектами зручності використання.
    - Підтримуваність (Maintainability): Програма повинна бути легкою для модифікації, виправлення помилок та розширення функціональності. Чітка структура коду, зрозумілі коментарі та модульність сприяють кращій підтримуваності.
2. Властивості програмної пентади: Програмна пентада розширює поняття основної пентади, включаючи аспекти, пов'язані з розробкою та життєвим циклом програмного забезпечення:

    - Функціональність (Functionality): Програма повинна надавати необхідний набір функцій для вирішення поставлених завдань. Функціональність визначається вимогами до програмного забезпечення.
    - Продуктивність (Performance): Ця властивість тісно пов'язана з ефективністю, але може також включати аспекти масштабованості та здатності програми обробляти великі обсяги даних або велику кількість користувачів.
    - Безпека (Security): Програма повинна бути захищена від несанкціонованого доступу, модифікації або знищення даних. Безпека включає заходи щодо аутентифікації, авторизації та захисту від вразливостей.

- Переносимість (Portability): Програма повинна мати можливість працювати на різних апаратних та програмних платформах без значних змін.
- Вартість (Cost): Вартість розробки, впровадження та підтримки програмного забезпечення є важливим фактором. Оптимізація вартості при збереженні якості є ключовим завданням.

3. Що таке часткова коректність програм? Яким чином доводиться часткова коректність програм?
   Часткова коректність програм означає, що якщо програма завершує своє виконання для певних вхідних даних, то результат її виконання відповідає специфікаціям. Іншими словами, якщо програма видає якийсь результат, то цей результат є правильним. Часткова коректність не гарантує, що програма обов'язково завершить свою роботу.

   Доведення часткової коректності програм зазвичай здійснюється за допомогою формальних методів, таких як:

   - Інваріанти циклу (Loop Invariants): Визначається логічне твердження (інваріант), яке є істинним перед початком циклу, залишається істинним після кожної ітерації циклу і використовується для доведення властивостей після завершення циклу.
   - Передумови та післяумови (Preconditions and Postconditions): Задаються умови, які повинні бути істинними перед виконанням певної частини програми (передумови), і умови, які повинні бути істинними після її виконання (післяумови). Доведення часткової коректності полягає в тому, щоб показати, що якщо передумови виконуються перед виконанням, і виконання завершується, то післяумови також виконуються.
   - Аксіоматичний підхід (Axiomatic Semantics): Використовуються формальні правила виведення для доведення тверджень про стан програми до і після виконання її фрагментів. Логіка Гоара є одним з прикладів аксіоматичного підходу.

4. Що таке повна коректність програм? Яким чином доводиться повна коректність програм?
   Повна коректність програм означає, що програма є частково коректною і, крім того, завжди завершує своє виконання для всіх допустимих вхідних даних. Тобто, для будь-яких вхідних даних, що задовольняють передумови, програма не тільки дасть правильний результат (відповідно до післяумов), але й обов'язково зупиниться.

Доведення повної коректності програм включає два етапи:

- Доведення часткової коректності: Як описано у попередньому пункті, використовуються інваріанти циклу, передумови та післяумови або аксіоматичні методи для показу, що якщо програма завершується, то результат є правильним.
- Доведення завершуваності (Termination): Необхідно показати, що програма (особливо її цикли та рекурсивні виклики) завжди досягає кінця виконання за скінченну кількість кроків. Для цього часто використовуються:
    - Функції спадання (Decreasing Functions): Для кожного циклу або рекурсивного виклику визначається функція, значення якої є цілим невід'ємним числом і строго зменшується з кожною ітерацією або рекурсивним викликом. Оскільки значення не може зменшуватися нескінченно, це гарантує завершення процесу.

5. Розкрийте зміст формалізації поняття програми.
   Формалізація поняття програми полягає у представленні програми як математичного об'єкта, що дозволяє аналізувати її властивості за допомогою строгих математичних методів. Це включає:

    - Визначення синтаксису мови програмування: Опис формальних правил побудови коректних програмних конструкцій (наприклад, за допомогою формальних граматик, таких як БНФ).
    - Визначення семантики мови програмування: Опис значення та поведінки програмних конструкцій. Існують різні підходи до формалізації семантики, включаючи:
        - Операційна семантика: Описує виконання програми як послідовність кроків обчислень на абстрактній машині.
        - Денсаційна семантика: Призначає кожній програмній конструкції математичний об'єкт (наприклад, функцію), що відображає її значення.
        - Аксіоматична семантика: Визначає набір логічних правил (аксіом та правил виведення) для доведення властивостей програм.
    - Представлення стану програми: Формальне визначення даних, з якими оперує програма (наприклад, змінні та їх значення).
    - Визначення процесу виконання: Математичний опис того, як програма змінює свій стан під час виконання.

6. Формалізація поняття програми є основою для розробки компіляторів, верифікації програмного забезпечення та теоретичних досліджень у галузі програмування.

7. Визначте різні класи функцій.
   У теорії обчислюваності та програмуванні розрізняють різні класи функцій залежно від їхніх властивостей та можливості обчислення:

   - Частково рекурсивні функції: Функції, які можуть бути обчислені за допомогою машини Тьюринга (або еквівалентної моделі обчислень), але можуть бути не визначені для деяких вхідних даних (тобто, процес обчислення може не завершитися).
   - Загальнорекурсивні (тотально рекурсивні) функції: Частково рекурсивні функції, які визначені для всіх можливих вхідних даних (тобто, процес обчислення завжди завершується). Ці функції відповідають інтуїтивному поняттю алгоритму.
   - Примітивно рекурсивні функції: Більш вузький клас загальнорекурсивних функцій, які можуть бути побудовані з базових функцій (нуль, наступник, проекція) за допомогою операцій суперпозиції та примітивної рекурсії. Не всі загальнорекурсивні функції є примітивно рекурсивними (наприклад, функція Аккермана).
   - μ-рекурсивні функції: Клас функцій, що включає примітивно рекурсивні функції та операцію мінімізації (μ-оператор). μ-оператор дозволяє знаходити найменше число, що задовольняє певну умову, і може призводити до незавершення обчислень, тому μ-рекурсивні функції є еквівалентними частково рекурсивним функціям.
   - Обчислювані функції: Зазвичай цей термін використовується як синонім частково рекурсивних функцій, оскільки вони можуть бути обчислені за допомогою формальної моделі обчислень.
   - Необчислювані функції: Функції, для яких не існує алгоритму (машини Тьюринга або еквівалентної моделі), що може обчислити їх значення для всіх або деяких вхідних даних. Прикладом є функція, що визначає, чи зупиниться задана програма на заданих вхідних даних (проблема зупинки).

8. Визначте програмні системи різного рівня абстракції.
   Програмні системи можуть розглядатися на різних рівнях абстракції, що дозволяє описувати їх з різним ступенем деталізації:

   - Апаратний рівень (Hardware Level): Найнижчий рівень, що включає фізичні компоненти комп'ютера (процесор, пам'ять, пристрої

введення-виведення). Програмування на цьому рівні зазвичай здійснюється за допомогою машинних кодів або мікрокодів.
    - Рівень мікроархітектури (Microarchitecture Level): Описує внутрішню організацію процесора, включаючи конвеєризацію, кешування, систему команд. Програмування на цьому рівні є дуже специфічним для конкретної архітектури.
    - Рівень машинних команд (Machine Code Level): Представляє програму у вигляді послідовності інструкцій, які безпосередньо виконуються процесором. Кожна інструкція виконує елементарну операцію.
    - Рівень асемблера (Assembly Language Level): Символічне представлення машинних команд, що полегшує програмування порівняно з машинним кодом. Потребує трансляції в машинний код за допомогою асемблера.
    - Рівень мов програмування низького рівня (Low-Level Programming Languages): Мови, що надають програмісту прямий доступ до апаратних ресурсів (наприклад, C, C++). Дозволяють ефективно керувати пам'яттю та іншими системними ресурсами.
    - Рівень мов програмування високого рівня (High-Level Programming Languages): Мови, що є більш абстрактними та орієнтованими на розв'язання конкретних задач (наприклад, Python, Java, JavaScript). Приховують деталі апаратної реалізації та полегшують розробку складних програм.
    - Рівень спеціалізованих мов та інструментів (Domain-Specific Languages and Tools): Мови та інструменти, розроблені для конкретних предметних областей (наприклад, SQL для роботи з базами даних, MATLAB для наукових обчислень).
    - Рівень моделей та специфікацій (Modeling and Specification Level): Найвищий рівень абстракції, на якому програмна система описується за допомогою формальних моделей (наприклад, UML) або специфікацій, без прив'язки до конкретної реалізації.
9. Дайте визначення класу номінативних даних.
   Клас номінативних даних (Nominal Data) - це тип даних, який представляє собою іменовані категорії або мітки без будь-якого природного порядку чи числової значущості. Значення в цьому класі є якісними відмінностями, а не кількісними. Операції, які мають сенс для номінативних даних, включають перевірку на рівність і нерівність, а також підрахунок частоти появи кожної категорії.

Приклади номінативних даних включають:

- Кольори (червоний, синій, зелений)
- Стать (чоловіча, жіноча, інше)
- Країни (Україна, Польща, Німеччина)
- Типи продуктів (овочі, фрукти, молочні продукти)

10. Важливо відрізняти номінативні дані від порядкових (ordinal data), які мають природний порядок (наприклад, розміри одягу: малий, середній, великий), та кількісних (quantitative data), які мають числову значущість (наприклад, вік, вага).

11. Повний клас обчислюваних функцій над номінативними даними.
Повний клас обчислюваних функцій над номінативними даними включає всі функції, які можуть бути ефективно обчислені за допомогою алгоритмів, що приймають номінативні дані як вхідні та повертають номінативні дані (або інші типи даних) як вихідні. Оскільки номінативні дані за своєю суттю є категоріальними, обчислювані функції над ними часто включають операції, пов'язані з:

    - Порівнянням: Перевірка на рівність і нерівність номінативних значень.
    - Класифікацією та категоризацією: Призначення номінативних значень до певних категорій на основі заданих правил.
    - Пошуком та фільтрацією: Вибір номінативних даних, що відповідають певним критеріям.
    - Агрегацією: Підрахунок кількості або частоти певних номінативних значень.
    - Перетворенням: Відображення одних номінативних значень в інші на основі заданих правил (наприклад, кодування).
    - Логічними операціями: Комбінування умов на номінативних даних за допомогою логічних операторів (AND, OR, NOT).

12. Формально, клас обчислюваних функцій над номінативними даними може бути визначений в рамках моделі обчислень, яка здатна оперувати символьними даними. Наприклад, машина Тьюринга може бути адаптована для роботи з алфавітом, що представляє номінативні значення. Будь-яка функція, яку може обчислити така машина, належить до класу обчислюваних функцій над номінативними даними.

Важливо зазначити, що через відсутність природного порядку серед номінативних даних, арифметичні операції (додавання, віднімання тощо) зазвичай не мають сенсу для цього класу даних, якщо тільки не визначені

спеціальні відображення на числові або інші структури даних.