Лекція 1. Парадигми програмування

Парадигма програмування – це сукупність ідей і понять, що визначають стиль написання програм. Парадигма в першу чергу визначається базовою програмною одиницею (визначення - декларативне, функціональне програмування, дія - імперативне програмування, правило - продукційне програмування, діаграма переходів - автоматне програмування, та ін. сутності). Дуже часто парадигма програмування визначається набором інструментів програміста, а саме, мовою програмування й використовуваними бібліотеками. Багато сучасних мов програмування є мультипарадигменними, тобто допускають використання різних парадигм. Напр., мовою Сі, що не є об'єктно-орієнтованою, можна писати об'єктно-орієнтованим чином.

Основні парадигми програмування:

- Імперативне (процедурне) програмування.
- Структурне програмування.
- Функціональне програмування.
- Логічне програмування.
- Об'єктно-орієнтоване програмування:

1) програмування, засноване на класах;
2) програмування, засноване на прототипах;
3) субъектно-орієнтоване програмування.

Імперативне програмування – парадигма програмування, згідно з якою описується процес отримання результатів як послідовність інструкцій зміни стану програми. Подібно до того, як з допомогою наказового способу в мовознавстві перелічується послідовність дій, що необхідно виконати, імперативні програми є послідовністю операцій комп'ютеру для виконання.

Імперативні мови базуються на ідеї змінної, значення якої змінюється присвоєнням. Вони називаються імперативними (лат. *imperative* - наказовий), оскільки складаються із послідовностей команд, які звичайно містять присвоєння змінних <назва_змінної> = <вираз>, де вираз може посилатися на значення змінних присвоєних попередніми командами.

Структурне програмування – парадигма програмування, в основі якої лежить представлення програми у вигляді ієрархічної структури блоків. Вона запропонована в 1970-х роках голландським науковцем Дейкстрою, була розроблена та доповнена Ніклаусом Віртом.

Згідно з цією методологією будь-яка програма - це структура створена на основі трьох основних конструкцій:

- **послідовне виконання** – однократне виконання операцій в тому порядку, в якому вони записані в тексті програми.
- **розгалуження** – однократне виконання одної з двох чи декількох операцій в залежності від виконання певної заданої умови.
- **цикл** – багатократне виконання операції доти доки виконується задана умова (умова продовження циклу)

Кожна конструкція являє собою блок із одним входом і одним або

декількома виходами.

Блок **Слідування** передбачає лінійне виконання операторів програми.

Блок **Вибір** являє собою точку прийняття рішення про подальший перебіг виконання операторів програми. Вибір здійснюється однією із трьох структур:

- **if** (єдиний вибір)
- **if...else** (подвійний вибір)
- **switch** або **case** (множинний вибір)

Усі три структури при бажанні можна звести до однієї типу **if.**

Блок **Повторення** реалізується одним із трьох способів:

- структура **while**
- структура **do/while**
- структура **for**

Усі три структури можна звести до структури **while.**

Структурована програма складається із вищеназваних блоків за двома правилами: **пакетування** (вихід одного блоку з'єднується із входом наступного) і **вкладення** (бадові конструкції можуть бутиь вкладені одна в одну довільним чином).

Таким чином, структуровані програми містять всього сім типів керуючих структур, які з'єднуються всього двома способами.

Такі програми легко створюються і тестуються. Розробка програм займає менше часу. Програми більш прозорі і легко піддаються переробці.

Повторювані фрагменти програми (або не повторювані, але представляючі собою логічно цілісні обчислювальні блоки) можуть оформлюватися у вигляді **підпрограм** (процедур або функцій). У цьому випадку в тексті основної програми, замість поміщеного в підпрограму фрагмента, вставляється інструкція виклику підпрограми. При виконанні такої інструкції виконується викликана підпрограма, після чого виконання програми триває з інструкції, що слідує за командою виклику підпрограми.

Теорема про структурне програмування:

Ця теорема була сформульована італійськими математиками К.Бомом і Дж.Якобіні в 1966 році й говорить нам про те, як можна уникнути використання оператора переходу goto.

Будь-яку схему алгоритму можна представити у вигляді композиції вкладених блоків begin і end, умовних операторів if, then, else, циклів із передумовою (while) і можливо додаткових логічних змінних (прапорів).

Перелічимо деякі достоїнства структурного програмування:

1.Структурне програмування дозволяє значно скоротити число варіантів побудови програми по одній й тій же специфікації, що значно знижує складність програми й, що ще важливіше, полегшує розуміння її іншими розроблювачами.

2.У структурованих програмах логічно зв'язані оператори перебувають візуально ближче, а слабко зв'язані - далі, що дозволяє обходитися без блок-схем і інших графічних форм зображення алгоритмів (по суті, сама програма є власною блок-схемою).

3.Сильно спрощується процес тестування й налагодження структурованих

програм.

Функціональне програмування – парадигма програмування, у якій процес обчислення трактується як обчислення значень функцій у математичному розумінні останніх (на відміну від функцій як підпрограм у процедурному програмуванні). Протиставляється парадигмі імперативного програмування, яка описує процес обчислень як послідовність зміни станів (у значенні, подібному такому ж в теорії автоматів).

Функціональна програма являє собою визначення функцій. Функції визначаються через інші функції або рекурсивно - через себе. У процесі виконання програми, функції отримують параметри, обчислюють і повертають результат, у разі необхідності обчислюючи значення інших функцій. Програмуючи на функціональній мові, програміст не повинен описувати порядок обчислень. Йому необхідно просто описати бажаний результат у вигляді системи функцій.

Майже кожен з нас так чи інакше використовував функціональний підхід до програмування. Візьмемо наприклад всім відомий Microsoft Excel. Записуючи вміст комірки у вигляді, схожому на звичайну математичну формулу, ми не замислюємося про дійсний порядк обчислень цієї формули, покладаючи ці функції на вбудований в Excel інтерпретатор.

Іншим простим прикладом можуть послужити мови запитів до баз даних. Мови засновані на реляційному обчисленні (наприклад всім відомий SQL або QBE) описують тільки результат, будучи тим самим по суті функціональними.

Програми на функціональних мовах звичайно набагато коротші й простіші, ніж ті ж самі програми на імперативних мовах.

Крім спрощення аналізу програм є ще одна вагома перевага - паралелізм. Раз всі функції для обчислень використовують тільки свої параметри, ми можемо обчислювати незалежні функції в довільному порядку або, скажемо, паралельно, на результат обчислень це не вплине. Причому паралелізм цей може бути організований не тільки на рівні компілятора з мови, але й на рівні архітектури.

Найбільш відомими мовами функціонального програмування є мови:

- LISP – вважається першою функціональною мовою програмування.
- Haskell – функціональна мова програмування. Названа на честь Хаскелла Каррі. Останній стандарт мови - Haskell 98.
- Erlang – (Joe Armstrong, 1986) функціональна мова з підтримкою процесів.
- F# – функціональна мова для платформи .NET.

ФП у нефункціональних мовах

У мові C покажчики на функцію можуть бути використані для одержання ефекту функцій вищого порядку. Функції вищого порядку й відкладена спискова структура реалізовані в бібліотеках C++. У мові C# версії 3.0 і вище можна використовувати λ-функції для написання програми у функціональному стилі.

Логічне програмування – парадигма програмування, заснована на виведенні з початкових фактів нових фактів згідно заданим логічним правилам.

Логічне програмування засноване на теорії математичної логіки. Найвідомішою мовою логічного програмування є Prolog, що є за своєю суттю універсальною машиною виводу, що працює в припущенні замкнутості системи фактів.

Об'єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Сьогодні багато мов програмування (зокрема, Java, C#, C++, Python, PHP, Ruby та Objective-C, ActionScript 3) підтримують ООП.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожний об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Клас. Клас визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її **атрибути** або **властивості**) та дії, які вона здатна виконувати (її **поведінку, методи** або **можливості**). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Як правило, клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значити, що клас повинний мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його **членами.**

Об'єкт. Окремий *екземпляр* класу. Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є **екземпляром (примірником**) класу Собака. Сукупність значень атрибутів окремого об'єкта називається **станом.**

Метод. Можливості об'єкта. Оскільки Сірко – Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). У межах програми, використання методу має впливати лише один об'єкт; всі Собаки можуть гавкати, алі треба щоб гавкала лише одна окрема собака.