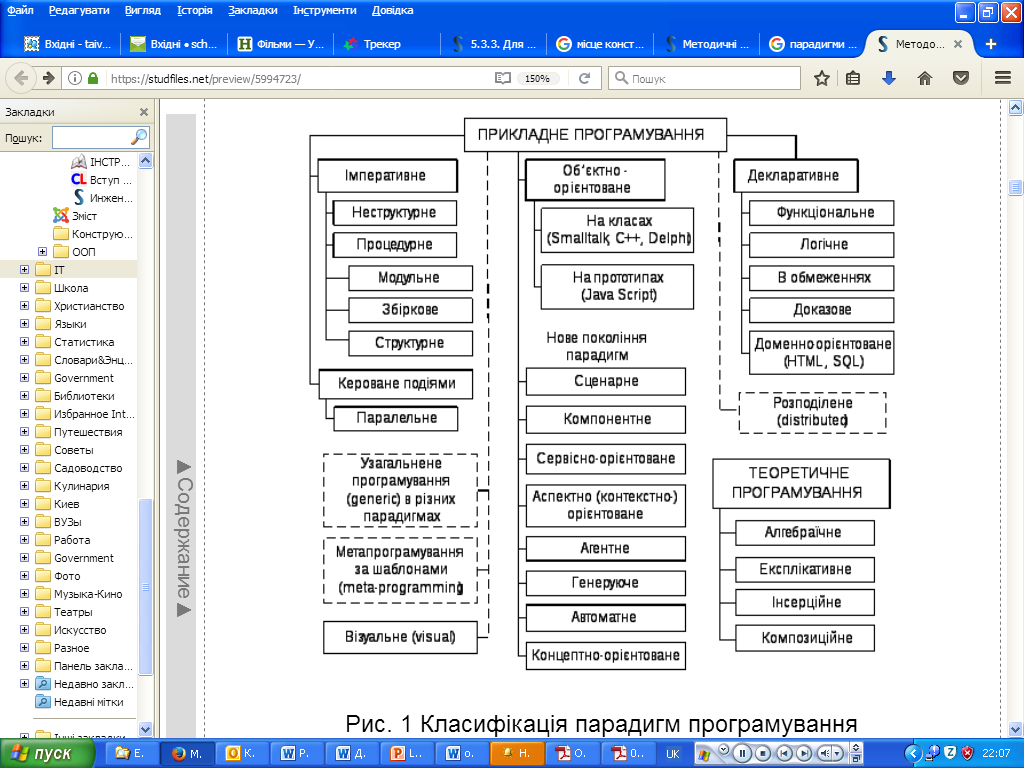
**Лекція 6. Вступ. Парадигми і мови програмування. Структурний та об’єктно-орієнтований підходи у програмуванні**.

Парадигми програмування

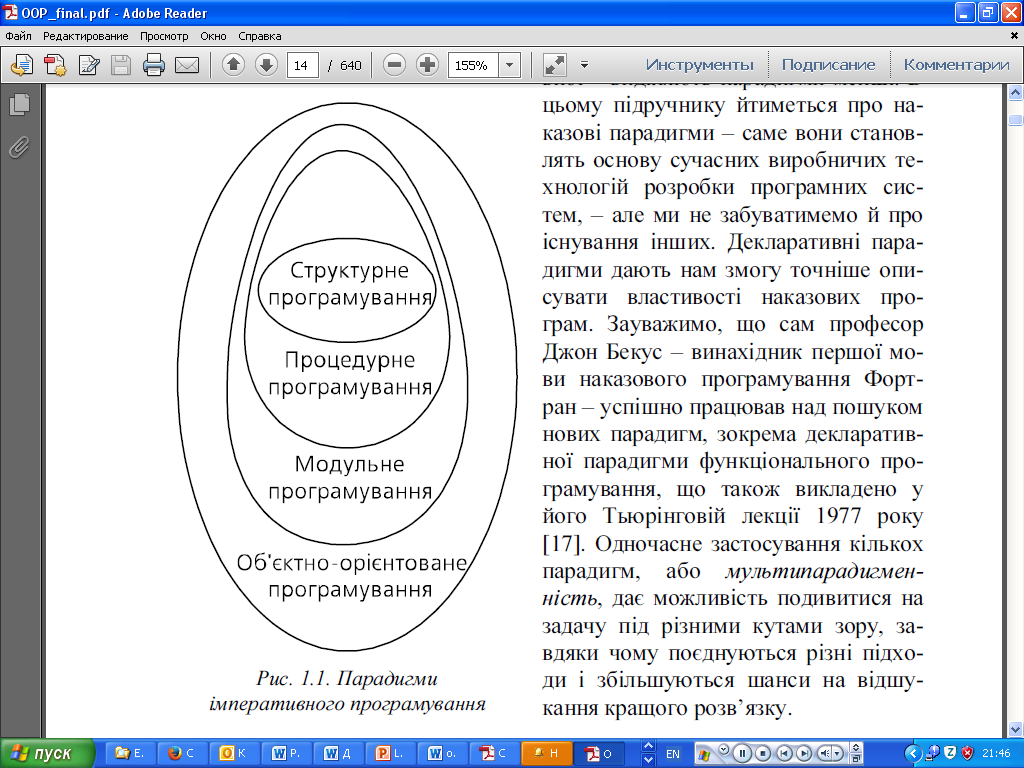
Слово парадигма грецького походження і означає стиль міркування,спосіб дій або набір концепцій у певній галузі знань. Парадигми програмування – це моделі, які відтворюють спосіб мислення розробника програми. Мова програмування може підтримувати або не підтримувати ту чи іншу парадигму. В першому випадку застосування парадигми стає зручним, тобто простим, безпечним і ефективним, в іншому – складним і ненадійним.

***Паради́гма програмува́ння***— це підхід до побудови програм, в основі якого лежить використання концептуально узгодженої сукупності ідей і понять, що визначають певний стиль написання програми.

Кожна з парадигм програмування має свою концептуальну основу, використовує свій набір моделей і методів обчислень, включаючи структури даних і механізми управління, і з нею пов'язаний певний клас прикладних задач, які зручно вирішувати засобами даної парадигми.Натепер існує достатньо велика кількість парадигм програмування (рис. 1). Одні парадигми розвиваються незалежно, інші з'являються в результаті поєднання різних концептуальних ідей.

Основними серед існуючих парадигм програмування вважаються наступні:

* імперативне програмування;
* об'єктно-орієнтоване програмування;
* декларативне програмування.



**Імперативне програмування –** це підхід до побудови програм, що використовує алгоритмічну декомпозицію задачі, при якій програма представляється як послідовність дій (команд), які повинен виконати комп'ютер покроковим чином.

Концептуальною ідеєю імперативного підходу до розробки програм є *алгоритмічна* структура програмного коду*, яка передбачає роз'єднання у програмах даних і дій, що виконуються над ними.* Активним суб’єктом у імперативних програмах вважається алгоритм, який повинен виконати всі необхідні для досягнення потрібного результату дії над пасивними даними. Таким чином, наголос у імперативному програмуванні робиться на верховенстві алгоритму над даними.

Імперативна парадигма програмування є відображенням архітектури традиційних ЕОМ, яка була запропонована фон Нейманом. Теоретичною основою імперативногопрограмування служить алгоритмічна система під назвою «машина Т’юринга» - абстрактний обчислювальний пристрій, що виконує послідовність команд програми, яка, таким чином, переходить з одного стану в інший. Відповідно програма в імперативному програмуванні розглядається як послідовність дій, які змінюють стан комп‘ютера.

Якщо розглядати стан комп'ютера як стан пам'яті, тобто значень комірок пам'яті, то імперативна програма – це послідовність операторів, які перетворюють вихідний стан пам'яті, тобто значення вихідних даних, у заключний, тобто у результати. Концепції пам'яті як сховища значень, поточного кроку виконання і поточного стану, що змінюється у часі, є фундаментальними в імперативному програмуванні.

Моделлю обробки даних при імперативному підході є послідовне виконання команд, які задають алгоритм вирішення задачі. Коротко це можна представити наступною нотацією: **програма = послідовність дій**.

Базовими поняттями імперативної парадигми програмування є поняття *оператора* (команди), що задає дію по обробці даних, та поняття *змінної*, якій може бути присвоєно значення, що зберігається в пам‘яті комп'ютера; базовою операцією є операція присвоєння, що служить для зміни вмісту областей пам'яті.

Основна задача програміста при написанні імперативної програми – звести рішення задачі до послідовності операторів, які вміє виконувати процесор комп'ютера.

Кожна парадигма програмування має відповідну інструментальну підтримку. Найбільш відомі і поширені імперативні мови програмування Фортран(1954), Алгол (1960), Паскаль(1970), С(1972).

Оскільки практично всі сучасні комп'ютери орієнтовані на послідовні обчислення, імперативне програмування явно виграє в ефективності реалізації прикладних задач, для яких важлива швидкість виконання. Окрім цього, робота із зовнішніми пристроями, як правило, описується в термінах послідовного виконання операцій, що робить такі задачі ідеальними для імперативної реалізації.

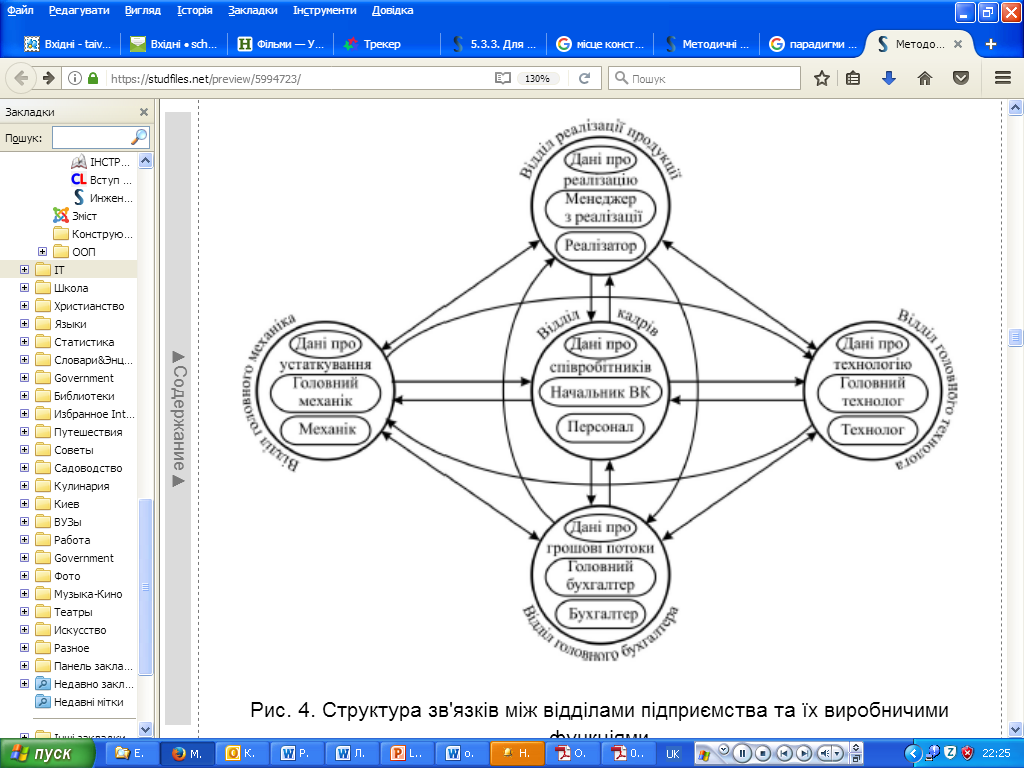
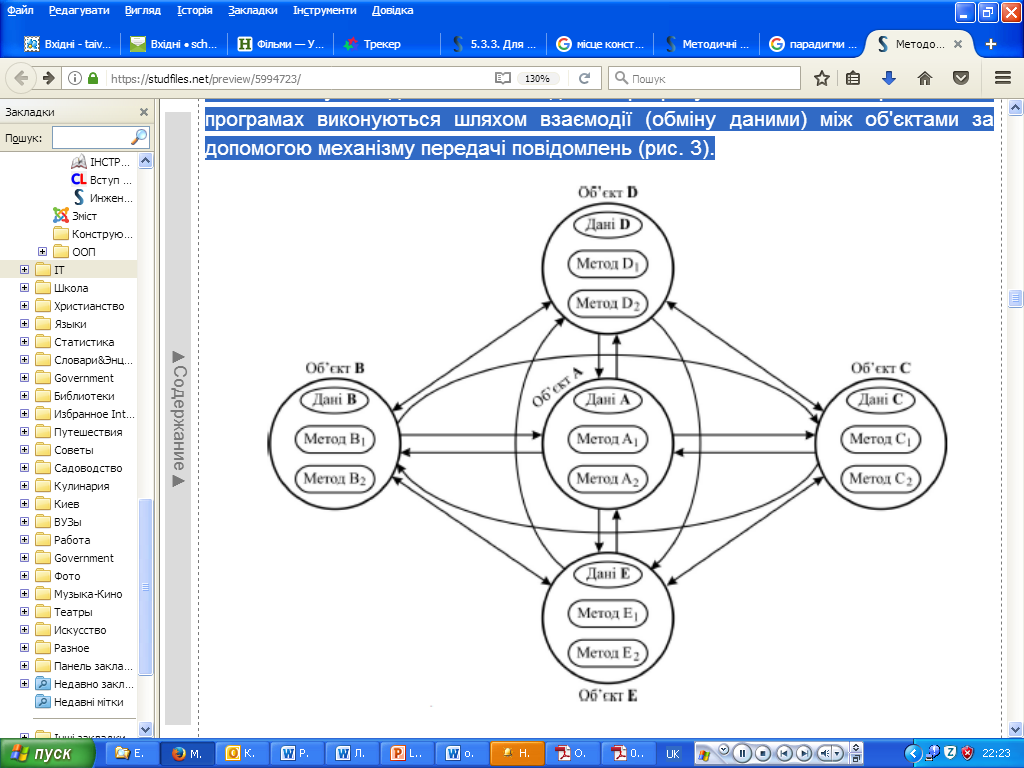
**Об'єктно-орієнтоване програмування** – це підхід до побудови програм, що використовує об'єктну декомпозицію задачі, при якій структура системи описується в термінах об'єктів і зв'язків між ними, а поведінка - в термінах обміну повідомленнями між об'єктами.

Об'єкт представляє собою цілісну модель і природну імітацію діяльності деякого елемента реального світу. Об'єкти, як і реальні сутності навколишнього світу характеризуються тим, що вони мають певний набір властивостей, здатні різними способами змінювати ці властивості і можуть реагувати на події, які виникають як у навколишньому світі, так і усередині самого об'єкта. Об'єкт, як правило, включає деякі дані, які характеризують стан об'єкта, та функції обробки цих даних, які описують поведінку об'єкта. Функції об'єкта, які у об'єктно-орієнтованому програмуванні (ООП) називають *методами*, зазвичай призначені для доступу до даних об'єкта та виконання певних дій над ними.

Поєднання даних і дій, що виконуються над ними, в єдине ціле, яке називають *об'єктом*, і є концептуальною ідеєю ООП.

Кожен об'єкт є екземп*ляром* певного *класу.* Клас в ООП - це в чистому вигляді *абстрактний тип даних*, що створюється програмістом. Клас представляє собою множину об'єктів зі схожою структурою і схожою поведінкою. *Усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.* Поняття *об'єкта* та*класу* є фундаментальними в ООП.

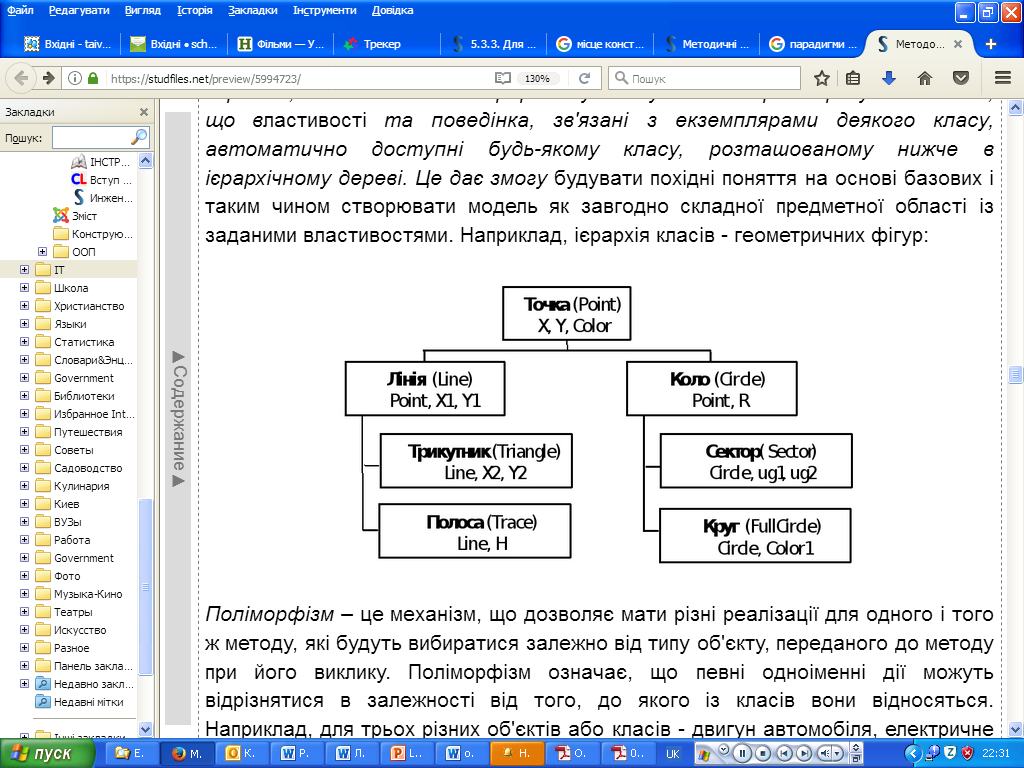
Об’єкти взаємодіють між собою шляхом надсилання *повідомлень*. Об'єкт, що приймає повідомлення, реагує на нього особливим, тільки йому відомим чином. При цьому він може послати повідомлення іншим об'єктам, отримати від них відповіді, змінити свій стан і, нарешті, повернути відповідь тому об'єкту, який послав йому повідомлення. Всі дії та розрахунки в об'єктно-орієнтованих програмах виконуються шляхом взаємодії (обміну даними) між об'єктами за допомогою механізму передачі повідомлень (рис. 3,4).



ООП ніяк не пов'язане з процесом виконання програми, а є тільки способом її ефективної організації. Основою такої організації є використання принципів *інкапсуляції*, *успадкування*та *поліморфізму*.

*Інкапсуляція* визначає спосіб опису, який передбачає об'єднання в межах класу даних і методів їхньої обробки, а також приховування деталей реалізації класу: користувач повинен бачити і використовувати для доступу до об'єкту і маніпулювання ним виключно інтерфейсну частину класу (список декларованих властивостей і методів). Це дозволяє забезпечити захист даних від зовнішнього втручання.

***Успадкування*** - це механізм породження нових класів з уже існуючих. Породжений клас успадковує властивості та поведінку класу-предка і може доповнити їх своїми власними. Механізм успадкування дозволяє *організовувати класи у єдину деревоподібну структуру із загальним коренем, яка називається ієрархією успадкування і характеризуються тим, що в*ластивості *та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві. Це дає змогу* будувати похідні поняття на основі базових і таким чином створювати модель як завгодно складної предметної області із заданими властивостями. Наприклад, ієрархія класів - геометричних фігур:



*Поліморфізм* – це механізм, що дозволяє мати різні реалізації для одного і того ж методу, які будуть вибиратися залежно від типу об'єкту, переданого до методу при його виклику. Поліморфізм означає, що певні одноіменні дії можуть відрізнятися в залежності від того, до якого із класів вони відносяться. Наприклад, для трьох різних об'єктів або класів - двигун автомобіля, електричне світло в кімнаті і комп'ютер, можна визначити операцію "вимкнути". Проте сутність цієї операції буде відрізнятися для кожного із розглянутих об'єктів. Так для двигуна автомобіля виклик дії "вимкнути двигун" означає припинення подачі палива і його зупинку. Виклик дії "вимкнути світло" означає просте клацання вимикача, після чого кімната зануриться в темряву. В останньому випадку дія "вимкнути компьютер" може бути причиною втрати даних, якщо виконується нерегламентованим чином. Використання поліморфізму робить програмне забезпечення більш гнучким і універсальним.

Таким чином, на відміну від імперативного програмування, що базується на алгоритмічній декомпозиції програми, а також роз'єднанні у програмах даних і процедур їхньої обробки, у об'єктно́-орієнтованих програмах має місце об'єктна декомпозиція програми і об’єднання даних та функцій їхньої обробки в межах певного класу. Первинними в ООП є дані (об'єкти), а не код. Об'є́ктно-орієнто́вана програма фактично є описом структури і поведінки окремих об'єктів, які взаємодіють між собою таким чином, щоб забезпечити певну поведінку системи.

Стійкість та керованість об'єктно-орієнтованої програми забезпечуються за рахунок чіткого розподілу відповідальності об’єктів (за кожну дію відповідає певний об’єкт), однозначного означення інтерфейсів міжоб’єктної взаємодії та повної ізольованості внутрішньої структури об’єкта від зовнішнього середовища (інкапсуляції).

Обчислювальна модель чистого об'єктно-орієнтованого програмування підтримує явно тільки одну операцію, якою є посилка об'єкту повідомлення. Повідомлення можуть мати параметри, які є об'єктами. Саме повідомлення також є об'єктом.

Роль програміста при написанні об'єктно́-орієнтованої програми полягає у формуванні і реалізації такої ієрархії об'єктів, взаємодія яких після запуску програми приведе до досягнення необхідного кінцевого результату. Використання раніше розроблених (можливо, іншими колективами програмістів) бібліотек об'єктів дозволяє значно заощадити трудовитрати при розробці програмного забезпечення, особливо типового.

Слід зауважити, що переваги ООП повною мірою виявляються лише при розробці досить великих програмних систем. Особливо зручно і легко в об'єктах виразитивзаємодіюміж різними елементами графічного інтерфейсу користувача. Спроби використовувати ООП для програмування нескладних алгоритмів, пов'язаних, наприклад, з розрахунками за готовими формулами, найчастіше виглядають штучними нагромадженнями непотрібних мовних конструкцій. Їх простіше і зручніше розробляти, застосовуючи імперативне програмування.

Об'єктно-орієнтовані мови можна розділити на три групи

* *Чисті мови*, які в найбільш класичному вигляді підтримують тільки одну об'єктно - орієнтовану парадигму програмування. Такі мови містять невелику мовну частину і істотну бібліотеку, а також набір засобів підтримки часу виконання. Приклади: Simula (1962), Smalltalk (1972)
* *Гібридні мови*, які з'явилися в результаті впровадження об'єктно-орієнтованих конструкцій в популярну імперативну мову програмування.C++(1983), ObjectPascal(1984)
* *Урізані мови,*які з'явилися в результаті видалення з гібридних мов найбільш небезпечних і непотрібних з об'єктно-орієнтованої точки зору конструкцій.С#(2000), Java(1995)

**Декларативне програмування**—це парадигма програмування, відповідно до якої програма деяким чином описує ***що*** потрібно отримати як результат, а не ***як*** це треба зробити. Тому декларативне програмування часто називають*описовим*.Тут головним є чітке формулювання метиі результатівроботи задачі, а непослідовністьотримання цього результату. Вибір і застосування необхідного для вирішення задачі алгоритму - проблема виконуючої системи (алгоритм роботи з даними "зашитий" в неї).

Так, декларативним є опис web-сторінок на HTML — вони описують, що містить сторінка та *що* має відображатись (заголовок, шрифт, текст, зображення), але не містять інструкцій *як* її слід відображати; опис SQL-запитів, які конкретизують властивості даних, які слід отримати від бази даних, але не процес отримання цих даних; описXML-документів тощо.

Такий підхід має високий ступінь абстракції і легко формалізується математичними засобами. Фактично програміст оперує не набором інструкцій, а абстрактними поняттями, часто досить узагальненими.

Декларативні мови найкраще використовувати у випадках, коли "дані управляють програмою": при написанні експертних систем, при конструюванні трансляторів з мов програмування, для більшості задач штучного інтелекту. Саме там їх використання призводить до найбільшої ефективності.

Різновидами декларативного програмування є *функціональне*та *логічне програмування*.

**Функціональне програмування**- це підхід до побудови програм, який грунтується на абстракції математичної функції і передбачає, що програма є сукупністю визначень математичних функцій, а її виконання полягає в обчисленні відповідних виразів.

При цьому функції можуть визначатися через інші функції (як *композиція* функцій) або рекурсивно (через самих себе). У процесі виконання програми функції отримують параметри, обчислюють і повертають результат, у разі необхідності обчислюючи значення інших функцій.

Слід зазначити, що існують відмінності в розумінні математичної функції у функціональному програмуванні і функції в процедурному програмуванні. Зокрема, функція в математиці не може змінити викликаючого її оточення і запам'ятати результати своєї роботи, а тільки надає результат обчислення функції. Тобто, у функційному програмуванні програму можна представити як обчислення послідовності функцій без станів.

На відміну від імперативного програмування, де існує поняття поточного кроку виконання і поточного стану, що змінюється у часі, у функціональному програмуванні поняття часу відсутнє. Ще однією особливість функціональних мов є їх безтиповість (відсутність типів даних).

Таким чином, функціональне програмування є способом створення програм, в яких єдиною дією є виклик функції, єдиним способом розбиття програми є створення нового імені функції та задання для цього імені виразу, що обчислює значення функції, а єдиною операцією, що використовується при визначенні функцій, - композиція функцій. Такий підхід дає можливість прозорого моделювання тексту програм математичними засобами. Основна специфіка функціональних мов програмування полягає в тому, що функції обмінюються між собою даними безпосередньо, тобто без використання проміжних змінних і присвоювань.

До відомих функціональних мов програмування належить LISP (*List Processing*), що розглядається фахівцями як основна мова програмування систем штучного інтелекту, Mathematica (символьні обчислення), XSLT (XML), Haskell, Scheme та ін.

Функціональне програмування застосовується для вирішення задач, які важко сформулювати в *термінах* послідовних операцій – переважно задач, пов'язаних з розпізнаванням образів, спілкуванням на *природній* мові, реалізацією експертних систем, автоматизованим доведенням теорем, символьними обчисленнями, тобто задач, які традиційно відносять до області штучного інтелекту.

**Логічне програмування** - це підхід до побудови програм, який грунтується на *логіці предикатів* і передбачає, що програма являє собою опис відомостей про задачу (*фактів*), припущень, достатніх для її вирішення (*правилвиведення*), і твердження, яке потрібно довести. Виконання програми полягає у доведенні цього твердження.

Факти (аксіоми) і правила виведення, які утворюють базу знань певної предметної області, виражаються в термінах предикатів. Твердження, яке вимагається довести, вводиться в програму як цільова функція. **Виконання** програми **полягає у** тому, щоб визначити за допомогою механізмів *логічного виведення*, чи витікає задане цільове твердження (запит) з наявних фактів і правил. Для виконання програми (логічного виведення результату) використовується вбудована система автоматичного пошуку.

Таким чином, у логічній програмі має місце умовний поділ на дані (факти) і код (правила, запити). Але цей поділ досить умовний: факти, правила і запити мають одну і ту ж форму запису – у вигляді предикатної функції.

Також слід зазначити, що програма на логічній мові програмування описуєнеалгоритм вирішення задачі, а логічну модель предметної області – деякі факти щодо властивостей предметної області і відношенняміж цими властивостями, а також правилавиведення нових властивостей і відношеньз уже заданих. Рішення задачі (отриманнявідповідіна запит) здійснються не шляхом виконання команд, а за допомогою механізмів керованогологічного виведення нових фактів на основі заданих фактів і правил виводу, які реалізуються програмною системою автоматично.

Основне завдання програміста - вдало описати у вигляді системи логічних формул предметну область і таку множину відношень на ній, які з достатнім ступенем повноти описують задачу.

Найбільш поширеною мовою логічного програмування є мова PROLOG (*PROgramming in LOGic* - програмування в термінах логіки).

Логічні програми відрізняються принципово низькою швидкодією, так як обчислення здійснюються методом проб і помилок (за допомогою пошуку з поверненнями). Однак, важливою перевагою такого підходу є достатньо високий рівень машинної незалежності, а також можливість відкатів - повернення до попередньої підцілі при негативному результаті аналізу одного з варіантів в процесі пошуку рішення (скажімо, чергового ходу при грі в шахи), що позбавляє від необхідності пошуку вирішення шляхом повного перебору варіантів і збільшує ефективність реалізації.

Декларативні мови найкраще використовувати у випадках, коли "дані управляють програмою": при написанні експертних систем, при конструюванні трансляторів з мов програмування, для більшості задач штучного інтелекту. Саме там їх використання призводить до найбільшої ефективності.

Узагальнимо основні властивості розглянутих парадигм програмування:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Парадигма* | *Ключовий концепт* | *Програма* | *Викоиання програми* | *Результат* |
| **імперативна** | оператор | послідовність операторів | викоиання операторів | результуючий стан пам’яті |
| **об’єктно-орієнтована** | **об’єкт** | набір класів о**б’єктів** | обмін повідомленнями між о**б’єкт**ами | результуючий стан о**б’єктів** |
| **функціональна** | функція | набір функцій | обчислення функцій | значення головної функції |
| **логічна** | предикат | логічні формули | логічний доказ | результат доказу |

Слід відзначити, що парадигма програмування не визначається однозначно мовою програмування — багато сучасних мов програмування є *мультипарадигменними*, тобто допускають використання декількох різних парадигм програмування. Зокрема, мова C++ є універсальною мовою програмування високого рівня з підтримкою декількох сучасних парадигм програмування: процедурної, об'єктно-орієнтованої та узагальненої.

Розвиток парадигм пов'язаний з двома основними взаємозв'язаними причинами:

1. розширенням класу функціональних задач і пошуком найбільш ефективних методів їх вирішення;
2. зростанням складності програм і систем, зростанням вимог до їхньої якості та надійності.

Мистецтво програмування якраз і полягає в тому, щоб вибрати одну з мов, яка найкраще підходить для вирішення наявної задачі.

***Методологія програмування*** - це об'єднана єдиним філософським підходом сукупність методів, що застосовуються в процесі створення програм.

З кожною методологією можна пов'язати деякі характерні для неї атрибути, а саме:

* парадигму програмування, що визначає основне джерело ефективності методології;
* узгоджену, зв'язну множину методів,через які реалізується дана методологія;
* концепції(поняття,ідеї), що підтримуюють методи і дозволяють більш точно їх визначити.

***Метод*** – це сукупність теоретичних принципів і практичних прийомів, які потрібно застосувати для виконання певної задачі.

Так основними методами, через які реалізується імперативна парадигма програмування, є:

* *метод зміни станів*, що полягає в послідовному зміні станів пам’яті (змінних) і підтримується концепцією алгоритму;
* *метод управління потоком виконання*, який полягає в покроковому контролі управління і підтримується концепцією потоку виконання.

Самий ранній стиль розробки імперативних програм характеризувався *неструктурованістю* - код програми представлявся єдиним безперервним блоком; переходи до потрібних секцій програми виконувалися за допомогою операторів переходу (завичай,goto). Подібні проекти отримали назву BS-програм – абревіатура від "bowl of spaghetti" – блюдо спагеті, бо саме так виглядала готова програма при спробі зобразити всі переходи між її операторами.

Хорошими програмістами на той час вважалися ті, хто створював достатньо хитромудрі програми, які займали мінімальний об'єм пам'яті та швидко виконувалися. Це було цілком природно, враховуючи тодішні можливості обчислювальної техніки. Однак, такі програми було важко (якщо взагалі можливо) зрозуміти іншим фахівцям. Інколи навіть автори таких програм з плином часу з трудом розуміли власне творіння. Тому при розробці таких програм виникали проблеми забезпечення якості: утворення «спагетті-коду», складність його перевірки і тестування.

Із збільшенням складності задач розмір програм постійно збільшувався. Типовою стала ситуація, коли в різних місцях програми виникала необхідність у виконанні однієї і тієї ж послідовності дій. Такі дії могли бути досить складними і представлятися великими фрагментами програми. Для забезпечення компактності і підвищення наочності програм було запропоновано фрагменти програми, що повторюються, оформляти у вигляді спеціальних синтаксичних конструкцій – *підпрограм* (процедур або функцій), які за необхідності могли бути викликані для виконання з будь-якого місця основної програми. Методологія програмування, заснована на концепції виклику підпрограми, що є логічно завершеним фрагментом програми, отримала назву ***процедурного програмува́ння***.

Основними ідеями, на яких грунтується процедурне програмування, є:

* використання механізмів передачі параметрів і повертання результатів;
* розподіл областей дії змінних програми (локальні, глобальні),
* використання різних моделей пам'яті для зберігання змінних;
* блокова організація програм.

Основними методами, через які реалізується процедурне програмування, є:

* *метод процедурної декомпозиції*, який полягає у оформленні кожного логічно завершеного фрагменти програми у вигляді підпрограми, і грунтується на концепції виклику підпрограми;
* *метод блочності*, що розглядає програму як деяку ієрархічну структуру, яка складається з головної програми і множини вкладених блоків (підпрограм), і грунтується на концепції блокової організації програм;
* *метод локалізаціїї даних*, який полягає у видимості (доступності) даних, що обробляються у підпрограмі, лише в межах підпрограми, і грунтується на концепції розподілу областей дії змінних програми;
* *метод специфікації інтерфейсів*, який полягає у формалізованому описі входів, виходів і підпрограм;
* *механізм раннього зв’язування*, який полягає устатичному зв’язуванні підпрограм (на етапі компіляції).

Моделлю обробки даних, характерною для процедурного програмування, є послідовне виконання підпрограм. Коротко це можна представити наступною нотацією:

***програма = послідовність процедур, кожна з яких є послідовністю елементарних дій і викликів процедур***.

Використання процедурного підходу при розробці програм дозволяє зменшити об'єм програмного коду, виключивши його дублювання,зробити програми прозорішими, прискорити процес їх написання і тестування. Використання підпрограм дозволило розбивати великі задачі на підзадачі і таким чином спростило написання великих програм. Завдяки даному підходу програми набули більш впорядкованого характеру.

Наступним кроком розвитку програмування стало підвищення структурованості програм – ***структурне програмування***. Дана методологія - це сукупність методів проектування та написання програм за жорсткими правилами, дотримання яких підвищує продуктивність праці програмістів, поліпшує читабельність і полегшує процес тестування програм.

Витоки цієї методології сягають 70-х років XX століття, коли швидкими темпами почала зростати складність програм, і постала потреба у методах подолання такої складності. Вона ґрунтується на теоремі Бома-Якопіні (1966 р.), які довели, що будь-який виконуваний алгоритм може бути перетворений до структурованого вигляду, тобто такого виду, коли хід його виконання визначається тільки за допомогою трьох структур управління: послідовностей, розгалужень і циклів.

Методологія структурного програмування реалізується через методи:

* структурного кодування (структурування програм);
* низхідного проектування (спадного проектування, проектування «зверху-вниз»);
* блочного програмування;
* паралельного документування.

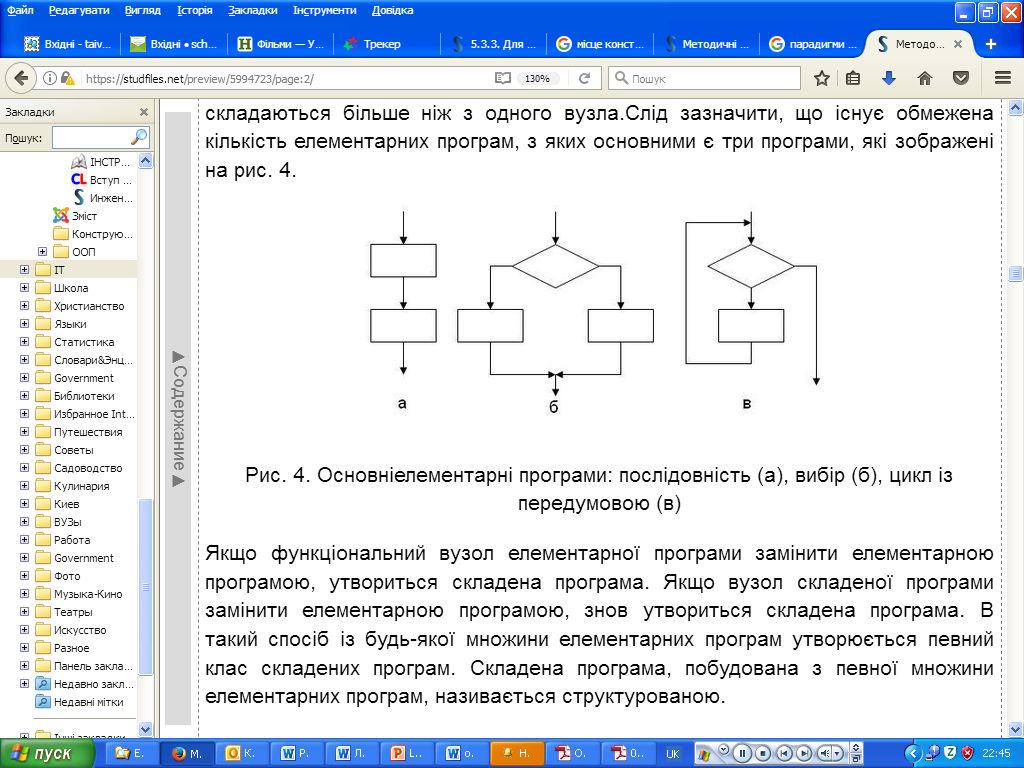
*Метод структурного кодування* передбачає, що будь-яка програма представляє собою структуру, побудовану із трьох типів базових конструкцій:

* *послідовне виконання* – одноразове виконання операцій в тому порядку, в якому вони записані в тексті програми;
* *розгалуження* – одноразове виконання однієї із двох або більшої кількості операцій, в залежності від виконання деякої заданої умови;
* *цикл* – багаторазове виконання однієї і тієї ж операції до тих пір, доки виконується деяка задана умова (умова продовження циклу).

В програмі базові конструкції можуть бути вкладені одна в одну довільним чином, але ніяких інших засобів керування послідовністю виконання операцій не передбачається. Зокрема, неприпустимим є використання оператора безумовного переходу (GOTO).

Метою структурування є перетворення неструктурованої програми на еквівалентну їй структуровану, тобто таку, що складається з обмеженого набору керуючих алгоритмічних структур. Методи структурування ґрунтуються на поняттях функціонального вузла, а також на поняттях простої, елементарної і складеної програми.

Елементарна програма — це проста програма, що не містить блоків, які складаються більше ніж з одного вузла. Слід зазначити, що існує обмежена кількість елементарних програм, з яких основними є три програми, які зображені на рис. 4.

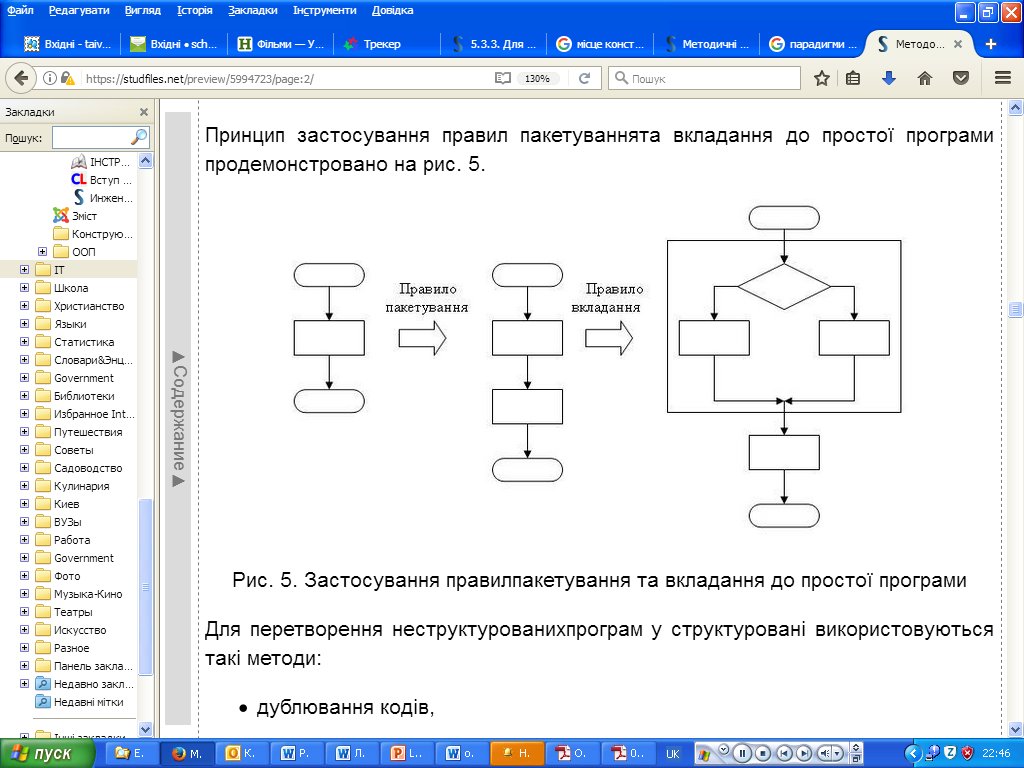


Якщо функціональний вузол елементарної програми замінити елементарною програмою, утвориться складена програма. Якщо вузол складеної програми замінити елементарною програмою, знов утвориться складена програма. В такий спосіб із будь-якої множини елементарних програм утворюється певний клас складених програм. Складена програма, побудована з певної множини елементарних програм, називається структурованою.

Позначимо літерою S клас складених програм, базисна множина якого містить такі елементарні програми, як послідовність (рис. 4, а), вибір (рис. 4, б) і цикл із передумовою ( рис. 4, в). Теорема про структурування стверджує, що будь-яку просту програму шляхом покрокового перетворення можна замінити функціонально еквівалентною структурованою програмою, що належить означеному вище класуS. Згадане покрокове перетворення здійснюється за переліченими нижче правилами:

1. *правило простоти* - створення програми слід починати із простої програми;
2. *правило пакетування* - кожну дію можна замінити двома послідовними діями (вихід одного функціонального вузла з'єднується із входом наступного);
3. *правило вкладання* –кожну дію можна замінити будь-якою структурою керування (будь-який блок може бути замінений на структуру керування вибору або повторення);
4. правила 2 і 3 можна застосовувати у будь-якій послідовності необмежену кількість разів.

Принцип застосування правил пакетуваннята вкладання до простої програми продемонстровано на рис. 5.



Для перетворення неструктурованих програм у структуровані використовуються такі методи:

* дублювання кодів,
* введення змінної стану,
* метод булевих ознак.

*Метод дублювання кодів* застосовується для перетворення неструктурованих програм, що не містять циклів. Для отримання структурованої програми дублюють ті блоки (модулі), у які можна увійти з декількох точок.

*Метод булевої ознаки* використовується для перетворення неструктурованих програм, що містять цикли. В програму вводиться додаткова змінна, яка набуває значення true чи false. Доки змінна ознаки зберігає це значення, виконання циклу триває. Значення змінної ознаки всередині циклу модифікується лише за певних умов.

*Блочне програмування* — це організація програми у вигляді сукупності відносно незалежних складових частин - блоків, якими фактично є підпрограми. Як показує практика, великі задачі простіше вирішуються, якщо розглядати їх як сукупність менших задач.

Як правило, програмний блок розв’язує порівняно нескладну задачу, логічно незалежну від інших задач. Його властивості:

* відповідає лише одній задачі;
* має один вхід і один вихід;
* має порівняно невеликий розмір;
* доступний за своїм ідентифікатором;
* може викликати інший блок (не в усіх мовах програмування);
* підтримує незалежність функціонування (заміна підпрограми на аналогічну не впливає на всю програму).

Тобто, кожна програма представляється лінійною послідовністю блоків. Кожен блок складається з одного або декількох інших блоків, кожен із яких має також рівно один вхід і рівно один вихід і сам може розглядатися як блок. Сенс блоків, що мають один вхід і один вихід, полягає в тому, що можна вийняти їх, змінити їх начинку і вставити назад без зміни інших сполучень в програмі.

Блоки повинні бути незалежними в межах інтерфейсу програми і структури даних. Практика показала, що чим вищий ступінь незалежності підпрограм, тим простіше розібратись в окремих програмних блокахі в програмі в цілому; тим менша ймовірність так званого *хвильового ефекту*– появи нових помилок при виправленні старих або при внесенні змін у програму. Тому не варто без крайньої необхідності використовувати в підпрограмах глобальні змінні. Всі зв'язки між підпрограмами мають підтримуватися через списки параметрів.

В основу *методу низхідного проектування* покладенос труктурну декомпозицію7задачі із застосуванням принципу покрокової (поетапної) деталізації. Спадне проектування передбачає, що розробка програм ведеться методом «зверху вниз», від загального до деталей. Спочатку задача розглядається як єдиний блок, що виражає загальне призначення програми. Далі ця задача поділяється на декілька дрібніших підзадач в тому порядку, в якому вони повинні виконуватися. Потім кожна з підзадач розбивається на свої підзадачі, що належать другому рівню деталізації і т.д. Процес покрокової деталізації підзадач здійснюється до тих пір, поки підзадачі чергового рівня не стануть досить простими для незалежного розв’язання (наприклад, кожній із них буде відповідати окрема команда мови програмування).

*Процедурне структурне програмува́ння* передбачає використання принципу *процедурної декомпозиції* на всіх рівнях проектування програмної системи. У відповідності із даним принципом кожен логічно цілісний програмний блок, отриманий на певному рівні деталізації, оформлюється як *підпрограма*. Наприклад, підпрограма обчислення визначника матриці, підпрограма знаходження суми елементів ряду тощо.

Розробка процедурної структурованої програми ведеться покроково, методом "зверху- вниз" з використанням механізму так званих «заглушок» - підпрограм, які нічого не роблять. При цьому спочатку пишеться основна програма, у якій замість кожного логічного зв'язного фрагмента тексту вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх, працюючих підпрограм, в програму вставляються "заглушки". Отримана програма перевіряється та налагоджується. Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності, тобто загальна структура програми вірна, підпрограми-"заглушки" послідовно замінюються на реально працюючі, причому розробка кожної підпрограми ведеться тим же методом, що і основної програми. Розробка закінчується тоді, коли не залишиться жодної "заглушки", яка не була б видалена.

Таким чином, «заглушки» дозволяють перевірити логіку програми верхнього рівня до реалізації наступного. Тобто на кожному кроці розробки програми існує працездатний «каркас», який поступово обростає деталями. Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступною для огляду і зрозумілою йому множиною фрагментів і може бути впевнений, що загальна структура всіх більш високих рівнів програми вірна. При супроводженні та внесенні змін у програму з'ясовується, в які саме процедури потрібно внести зміни, і вони вносяться, не зачіпаючи безпосередньо не пов'язані з ними частини програми. Це дозволяє гарантувати, що при внесенні змін і виправленні помилок не вийде з ладу якась частина програми, що знаходиться в даний момент поза зоною уваги програміста.

Таким чином, при структурному програмуванні програма ієрархічно структурується і розробляється шляхом послідовного уточнення на кожному рівні ієрархії. В основу цього процесу, окрім принципу ієрархічності, покладено принципи абстрагування.

*Абстрагування*— це спрощений опис системи, в якому зосереджують увагу на певних властивостях і деталях, а на інші не зважають. Вдалою є та абстракція, що підкреслює суттєві деталі і відкидає несуттєві. Під час низхідного проектування програми на верхніх рівнях абстракції деталі реалізації приховуються, а на нижніх рівнях вони описуються конкретною мовою програмування.

Фактично такий підхід збільшив структурованість програм – велика програма стала сукупністю процедур-підпрограм. Одна підпрограма, головна, розпочинала роботу всієї програми.

Одним із прийомів формалізованого підходу до низхідного процедурного проектування є метод ієрархічних діаграм, що позначається абревіатурою НІРО (від англ. *Hierarchical Input Processing Output* — діаграма входу, обробки, виходу). Згідно з цим методом структура всієї програми подається у вигляді дерева, в якому підпрограми зображуються вузлами, а їхні виклики — ребрами.

Методи низхідного проектування та блочного програмування регламентують процес побудови архітектури програм на макрорівні. Методи структурування, навпаки, працюють на мікрорівні, регламентуючи процес створення програмного коду.

Сутність методу *паралельного документування* полягає у тому, що при розробці програм документація повинна створюватися одночасно із програмуванням, зокрема, у вигляді коментарів до програми. Документування програмного тексту дозволяє будь-якому програмісту легко читати і розуміти його у процесі розробки та подальшого супроводу.

Впровадження принципів структурного програмування зробило тексти програм, навіть досить великих, більш читабельними. Серйозно полегшилось розуміння програм, з'явилася можливість розробки програм в нормальному промисловому режимі, коли програму може без особливих труднощів зрозуміти не тільки її автор, а й інші програмісти. Все це полегшило і прискорило процес розробки програмного забезпечення.

Розвитком ідей процедурного програмування, пов'язаним із зростанням об'єму і складності програмного забезпечення, стало так зване ***модульне програмування***. Основна ідея модульного програмува́ння полягає у організації програм як сукупності незалежних блоків, які прийнято іменувати *модулями*.

Визначальним принципом модульного програмування є незалежність модулів. Це означає, що програма має бути поділена на модулі таким чином, щоб зв'язки між модулями були слабкими, а зв'язки всередині модулів — сильними. Модулі вважаються незалежними, якщо вони задовольняють таким вимогам:

* кожен модуль можна замінити іншим функціонально еквівалентним модулем, що має той самий інтерфейс;
* взаємозв'язки між модулями встановлені за ієрархічним принципом;
* усі структури даних інкапсульовані в модулі, тобто доступ до даних може здійснюватися лише через процедури та функції модуля;
* модуль має одну точку входу та одну точку виходу;
* модуль повертає керування тому програмному блоку, що його викликав.

При розбитті програмного забезпечення на модулі для кожного модуля визначається функціональність, що реалізовується ним, а також зв'язки з іншими модулями. При цьому функціональність модуля (**що** він повинен виконувати) відокремлюється від його реалізації (**як** це виконується). Зв'язки між модулями здійснюються через спеціальний інтерфейс, тоді як доступ до реалізації модуля (тіл підпрограм і деяких «внутрішніх» змінних) заборонений. Тобто, знаючи, що робить модуль і як його викликати, можна без обмежень використовувати його, не маючи при цьому жодного уявлення, як саме він реалізований. Такий підхід розмежовує доступ до даних програми, зменшуючи кількість помилок, що виникають при роботі з ними.

Ще одна ідея модульного програмування полягає угрупуванні підпрограм і даних, якими вони управляють,в окремо компільовані файли, які також називають модулями. Основна программа може підключати такі модулі і використовувати підпрограми, що містяться в них. Тим самим забезпечується можливість автономної реалізації і зберігання коду (в бібліотеках модулів), а також збирання програм з модулів.

Основними ідеями, на яких грунтується модульне програмування, є:

* автономна реалізаціямодулів (в бібліотеках модулів),
* автономна компіляція,
* збирання програм з модулів.

Основними методами, через які реалізується модульне програмування, є:

* *метод незалежності модулів*, який полягає у розробці модулів, відносно ізольованих один від одного;
* *механізм пізнього зв’язування*, який полягає у динамічному виклику модулів (на етапі виконання).

Розбиття на модулі зменшує час перекомпіляції і полегшує процес налагодження програм, приховуючи несуттєві деталі за інтерфейсом модуля і дозволяючи налагоджувати програму по частинах. Інтерфейсом модуля є заголовки всіх підпрограм і описи доступних ззовні типів, змінних і констант. Використання модульного програмування істотно спрощує розробку програмного забезпечення, зокрема, шляхом розподілу процесу його розробки між групами розробників. Крім того, модулі надалі без змін можна використовувати в інших розробках, що підвищує продуктивність роботи програмістів. У разі колективної розробки модульної програми робота розподіляється так: більш досвідчені програмісти відповідають за інтерфейс модулів, а менш досвідчені — за їх реалізацію.

Модульний принцип покладено в основу практично всіх стилів програмування.

**Контрольні запитання.**

1. Надайте визначення парадигмі програмування та поясніть чим викликана поява різних парадигм.
2. Що називають програмним модулем?
3. Охарактеризуйте метод функціонального програмування.
4. Охарактеризуйте метод структурноного програмування.
5. Охарактеризуйте метод об’єктно-орієнтованого програмування та визначте його базові принципи.
6. Охарактеризуйте відмінності процедурного, об’єктного та об’єктно-орієнтованого програмування.
7. В чому полягає різниця між процедурним та модульним програмуванням?
8. Яка технологія пропонується для розробки процедурної структурованої програми?

**Для самостійного вивчення:** Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Література**

1. *Соммервилл Иан.* Инженерия программного обеспечения. — М.: Вильямс, 2002. — 624 с.
2. Бандоріна Л.М., Климкович Т.О., Удачина К.О. Основи алгоритмізації та програмування : навч. посібник. УДУНТ, 2022. 158 с. URL: <http://eadnurt.diit.edu.ua/bitstream/123456789/15729/1/Bandorina.pdf>
3. Боровльова С. Ю. Базовий С++ : навчальний посібник /С. Ю. Боровльова, А. В. Швед. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2017. – 116 с.. URL: https://dspace.chmnu.edu.ua/jspui/bitstream/123456789/56/1/%D0%91%D0%BE%D1%80%D0%BE%D0%B2%D0%BB%D1%8C%D0%BE%D0%B2%D0%B0%20%D0%A1.%20%D0%AE.%20%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D0%B8%D0%B9%20%D0%A1%2B%2B.pdf
4. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997. URL: http://khizha.dp.ua/library/Timothy\_Budd\_-\_Introduction\_to\_OOP\_(ru).pdf.

**Довідкова інформація:**

Електронна адреса викладача: [***t.i.lumpova@gmail.com***](mailto:t.i.lumpova@gmail.com)

+380683979566 Вайбер, Телеграм

063-1287-961 дзвонити протягом робочого дня – до 18-00

Посилання на сайт, де зберігаються лекції, лабораторні роботи та інша інформація щодо предмета "Основи програмування та алгоритмічні мови"

***https://github.com/TetyanaLumpova/OP-AM/tree/main/2024***