Декларативне програмування

Лектор

доц., к.т.н. Левус Є.В.

Керівник лабораторних занять

ст.викл., к.т.н. Тушницький Р.Б.

Декларативне програмування

=

Функційне програмування

+

Логічне програмування

Це 2 частини

Інша термінологія

- Функційне (функціональне) програмування =

  Аплікативне програмування
- Логічне програмування =

  Реляційне програмування

Мета вивчення дисципліни

- Ознайомитися з новою парадигмою програмування
- Вивчити засоби функційного та логічного програмування для вирішення прикладних та наукових задач

Мови програмування

1. Scheme (сімейство LISP/ЛІСП),

Середовище DrRacket, DrScheme

2. Prolog

Середовища Turbo Prolog (?),

Visual Prolog

Декларативне програмування (оцінювання)

- Екзамен (70 б)
- 8 лабораторних робіт (30 б)
- Модульний контроль по 35 балів

Лабораторні роботи

- До І модуля 4 роботи – 11 б (2+3+3+3)
- Після І модуля 4 роботи -19 б (4+5+4+6)
- 5 робіт до функційного програмування і 3 роботи до логічного програмування
- Функційне та логічне програмування базуються на одних і тих же теоретичних ідеях

Методичні матеріали

I частина

1. Левус Є.В. Елементи функційного програмування./Метод.вказівки до практ.робіт з дисципліни “Логічне і функційне програмування” Львів: - Нац. унів. “Львівська політехніка”: 2007.
2. Дехтяренко И.А. Декларативное программирование –2003.(Інтернет-ресурс)
3. Хювёнен Э., Сеппенен И. Мир Lisp'a. В 2-х томах. М.: Мир, 1990.
4. Заяць В.М. Функційне програмування/Посібник для ст.ВНЗ. – Л.:2002.

II частина

1. Братко И. Алгоритмы искусственного интелекта на языке PROLOG. – М.:2004.
2. Левус Є.В., Сердюк П.В. «Створення найпростіших програм у середовищі Visual Prolog»/ Метод.вказівки до виконання лабор. робіт– Львіів: Вид-во Львівської політехніки, 2010.
3. Заяць В.М. Конспект лекцій з курсу “Логічне програмування”.-Львів, 2002

Лекція 1.

Вступ до декларативного програмування

(2 год., самостійне вивчення)

Специфіка вивчення

- Це нетрадиційне програмування
- Орієнтувалося на комп`ютери не “фон-нейманівської архітектури”
- Принципово відрізняється від імперативного програмування

Формальне визначення ОМ за фон Нейманом

Обчислювальна машина є машиною з архітектурою фон-Неймана, якщо:

1. Програма та дані зберігаються в одній загальній пам'яті.
2. Кожна комірка пам'яті машини ідентифікується унікальним номером, який називається адресою.
3. Різні слова інформації (команди та дані) розрізняються за способом використання, але не за способом кодування та структурою представлення в пам'яті.
4. Кожна програма виконується послідовно, починаючи з першої команди, якщо немає спеціальних вказівок. Для зміни цієї послідовності використовуються команди передачі управління.

Недоліки архітектури фон Неймана

1. Семантичний розрив
2. Розділення операційного пристрою та пам'яті
3. Послідовний принцип виконання

Семантичний розрив

Примітивний та низькорівневий набір команд, який, на думку критиків, абсолютно не відповідає сучасному стану справ в індустрії розробки програмного забезпечення.

Тобто, потрібно зазвичай до декількох сот машинних команд замість однієї команди мови високого рівня.

У 60-70 роки XX століття було досить багато намагань реалізувати машинні мови високого рівня апаратно (Архітектура з розвинутими засобами інтерпретації).

Серед вітчизняних розробок в цьому напрямі слід виділити ЕОМ серії "МИР", а серед серйозних критиків системи фон-Неймана, в тому числі і за низький семантичний рівень команд, академіка В. М. Глушкова.

Розділення операційного пристрою та пам'яті

Це так зване "пляшкове горло" фон-нойманівської архітектури (термін, запропонований Джоном Бекусом (John Backus)в 1977).

Це "горло" створюється між центральним процесором і пам'яттю, адже *швидкість обробки інформації в процесорі зазвичай є набагато більшою, аніж швидкість роботи запам'ятовуючого пристрою, який не встигає забезпечувати процесор новими порціями інформації, що призводить до простоїв.*

Проблема вирішується за рахунок побудови більш складної ієрархії пам'яті, зокрема введенням кеш-пам'яті, більш швидкої (але й більш дорогої, аніж основна), де зберігаються дані, які часто використовуються в обчисленнях, щоб не звертатись за ними до повільної основної пам'яті.

Прикладом часткового вирішення цієї проблеми є гарвардська архітектура, в якій пам'ять команд та даних розділена, що дозволяє інтенсифікувати обмін між запам'ятовуючим пристроєм та центральним процесором.

Послідовний принцип виконання

І це є суттєвим обмежуючим фактором в підвищенні швидкодії машин, унеможливлює введення явного паралелізму в систему.

Передусім це питання не технічне, а концептуальне і пов'язане з самою парадигмою програмування для фон-нейманівських машин.

Хоча майже всі ЕОМ загального призначення є фон-нейманівськими, вони суттєво використовують механізми розпаралелення обчислень.

Це відбувається неявно, на рівні внутрішньої організації процесора, який непомітно для програміста виявляє схований паралелізм в послідовних програмах для фон-нейманівських машин.

Фактично фоннеймановською в сучасних ЕОМ залишається саме архітектура обчислювальної машини (**тобто програмна організація**).

Внутрішня організація сучасних процесорів радикально **використовує нефоннойманівські принципи виконання команд,** але відкриття їх для програміста насправді може зруйнувати всю індустрію, і саме в цьому є секрет привабливості фоннеймановської архітектури.

Фактично, ця концепція пропонує програмісту надзвичайно **просту модель виконання програми**, послідовну модель, яка співпадає з образом мислення більшості програмістів, яка є домінуючою в написанні програм.

Явне паралельне програмування - це надзвичайно складна галузь, яка потребує повної перебудови образу мислення програміста, оперування більш складними абстракціями, застосування зовсім інших алгоритмів та структур даних.

Характерні особливості програмування на комп’ютерах фон Неймана призводять до розподілу праці:

є люди, які думають як вирішити задачу, і розробляють відповідні методи – аналітики, а є кодувальники, які пишуть тексти програм, тобто виконують прозаїчну і рутинну роботу з перекладом інструкцій у команди.

Імперативне програмування

=

Процедурне програмування

+

Об`єктно-орієнтоване програмування

Програма, написана імперативними мовами, є послідовністю вказівок, відповіддю на запитання:

“Що треба зробити, щоб отримати результат?”

Імперативне програмування

- Спрямоване на пошук методу вирішення задачі (алгоритм, метод, пасивні дані, активні дані…)
- Це схоже до спонукального (наказового) стилю висловлювання

Декларативне програмування

- Спрямоване на опис (формалізовану постановку) задачі, а не на пошук вирішення задачі
- Вбудовані засоби в середовища програмування мають методи, алгоритми, які дають рішення для задачі. Програмістові не треба самому шукати метод вирішення задачі
- Це схоже до розповідного стилю висловлювання

Вище перераховані твердження щодо декларативного програмування приводять до висновку, що **декларативне програмування має легше сприйматися до вивчення, ніж імперативне**. Бо воно ближче до способу висловлювання звичайної мови звичайних людей

Насправді, стереотипи імперативного програмування настільки сильні,

що декларативне програмування вивчається не так легко.

Наприклад, можна уявити програмування без змінних, операторів присвоєння, циклів..?

Сучасні середовища розробки декларативних програм, звичайно ж, мають засоби ООП, візуального, процедурного програмування.

Проте ми їх не будемо використовувати, а будемо програмувати в строго декларативному стилі.

Історія розвитку мов програмування

в ракурсі машинної незалежності

І етап (машинні мови)

- 40-50-ті роки XX століття

Процес програмування передбачав запис програмістом усіх алгоритмів безпосередньо машинною мовою (двійкові та шістнадцяткові коди).

Реалізація алгоритмів таким способом була дуже незручною, складною та клопітливою. “Блукання” лабіринтами регістрів, адрес пам’яті, двійкових розрядів не дозволяли сконцентруватися на особливостях вирішуваної задачі.

II етап (мови асемблера)

- 50-60-ті роки XX століття

Мнемонічні записи різних команд замість шістнадцяткових кодів , а також ідентифікатори замість конкретних адрес комірок пам’яті.

Розроблені асемблери – програми для перекладу програм, записаних у мнемонічному вигляді, на машинну мову. Мнемонічні системи запису програм– мови асемблера.

Відмінності між мовами програмування I і II поколінь у синтаксисі (способі вираження конструкцій).

Мови асемблера є також машинно-залежні, тобто команди виражаються в термінах певних машинних атрибутів, враховується конфігурація регістрів та набору команд.

III етап (умовно машино-незалежні мови)

- 60-70-ті роки XX століття

Теоретично, програма, написана мовою програмування може бути виконана на будь-якій ЕОМ за рахунок використання відповідного компілятора.

У дійсності при розробці компілятора приходилося враховувати певні обмеження, які накладаються тою ЕОМ, для якої компілятор призначений. Ці обмеження, звичайно, відображаються у мові програмування. Наприклад, розмір значень цілих змінних і т.п.

Першими найбільш відомими такими мовами були FORTRAN і COBOL.

Розвивається ідея орієнтації на певні проблеми. Проблемно-орієнтовані мови програмування

IV покоління

- з 70-их рр XX століття

Мови програмування, які дозволяють спілкуватися з комп`ютером в термінах абстрактних понять, ближчих до людської мови.

Об`єктно-орієнтовані програмування і абстрактні типи даних – класи.

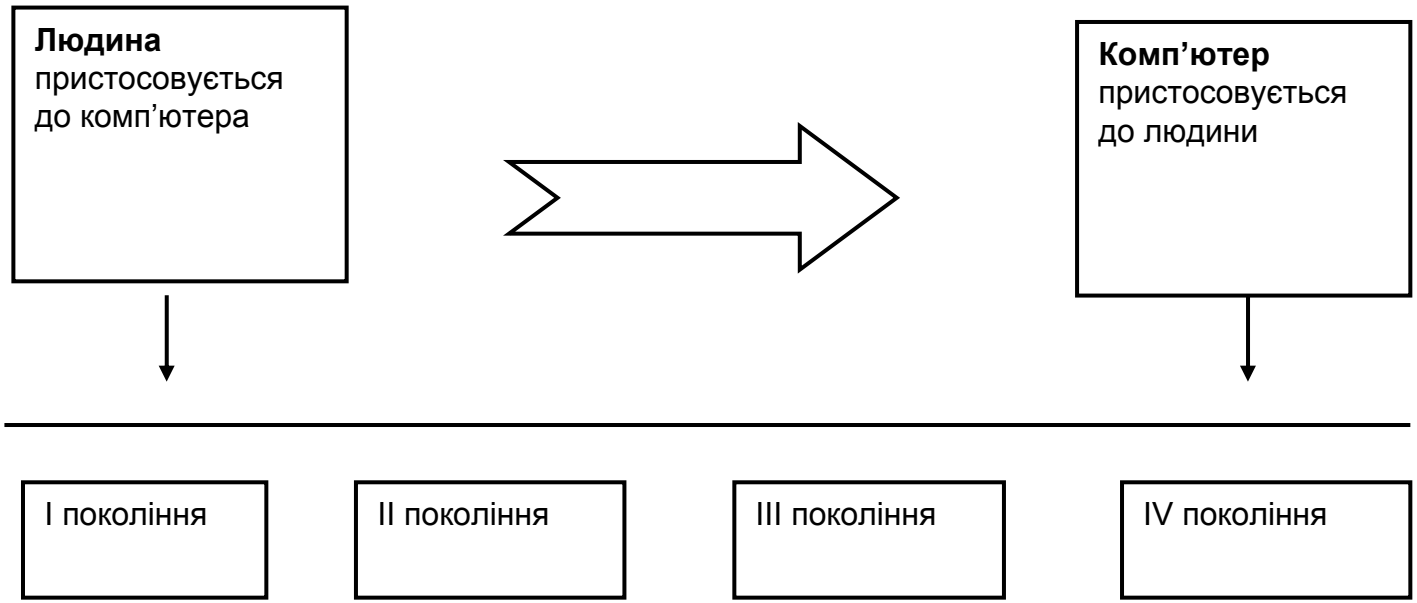
V покоління

Мета – створити мови програмування, що оперують термінами людської мови.

Близькі ідеї декларативних мов.

Декларативні мови належать і до IV, і частково до V покоління.

Тенденція розвитку мов програмування



Основна ідея декларативного програмування

Програма має базуватися на абстрактній специфікації проблем, а не бути описом методів їх вирішення.

- Витоки декларативного програмування лежать в математиці і логіці
- Декларативне програмування орієнтоване на символьну обробку даних, оскільки спочатку задумувалося саме для задач штучного інтелекту

Самостійне вивчення тем (домашнє завдання)

1. Класична структура машини фон Неймана
2. CISC- і RISC-архітектури системи команд: недоліки і переваги
3. Архітектура з розвинутими засобами інтерпретації
4. ЕОМ серії "МИР" як противага архітектурі фон Неймана
5. Штучний інтелект як розділ комп`ютингу