Лекція №1 «Поняття алгоритму та моделі. Історія розвитку мов програмування.»

Поняття інформаційної моделі

З давніх часів людина використовує моделювання для дослідження об'єктів, процесів та явищ в різних галузях своєї діяльності. Результати цих досліджень допомагають визначити та покращити характеристики реальних об'єктів та процесів, краще зрозуміти сутність явищ та пристосуватись до них або керувати ними, конструювати нові та модернізувати старі об'єкти. Моделювання допомагає людині приймати обґрунтовані рішення та передбачати наслідки своєї діяльності.

Поняття комп'ютерного моделювання відображає використання в цьому процесі комп'ютера, як потужного сучасного засобу обробки інформації. Завдяки комп'ютеру суттєво розширюються галузі застосування моделювання, а також забезпечується всебічний аналіз отриманих результатів.

Що ж таке модель? Під цим словом ховаються і матеріальні моделі реально існуючих об'єктів на виставці, і телевізійна красуня, що рекламує товари та сучасний одяг, і макет Ейфелевої вежі, і теорія розвитку суспільства, і всім відома формула земного тяжіння $P = mgh$, і багато чого іншого. Як же в одному слові можна об'єднати такі різні поняття?

Справа в тому, що поняття моделі об'єднує дещо спільне, а саме те, що модель - це штучно створений людиною абстрактний або матеріальний об'єкт. Аналіз та спостереження моделі дозволяє пізнати сутність реально існуючого складного об'єкта, процесу чи явища, що називаються прототипами об'єкта. Таким чином, модель - це спрощене уявлення про реальний об'єкт, процес чи явище, а моделювання - побудова моделей для дослідження та вивчення об'єктів, процесів та явищ.

Може виникнути питання, чому не можна дослідити сам оригінал, навіщо створювати моделі? Для цього може існувати багато причин:

1. оригінал може не існувати в часі (гіпотеза про загиблий материк Атлантида, про побудову Єгипетських пірамід, про можливу "ядерну зиму", що може початися після атомного бомбардування);
2. реально цей об'єкт не можна побачити (модель земної кулі, сонячної системи або атома);
3. людина хоче побачити об'єкт, але не має можливості потрапити на місце його знаходження (модель Ейфелевої вежі, єгипетської піраміди, Софіївського собору тощо);
4. процес, який досліджує вчений, небезпечний для життя (ядерна реакція).

Таких прикладів можна придумати багато.

Для одного і того ж об'єкта можна створити велику кількість моделей. Все залежить, по-перше, від мети, що ви поставили перед собою, а по-друге, від методів та засобів, за допомогою яких ви збираєте інформацію про прототип.

Наприклад, якщо ви хочете познайомитись з новим містом, то карта цього міста, фотографії, розповіді мешканців або кіноальманах дадуть вам зовсім різні уявлення про об'єкт, причому ці уявлення можуть зовсім не співпасти з тими враженнями, що ви отримаєте потім після відвідування цього міста безпосередньо. Модель цього ж самого міста для його мешканців взагалі буде іншою, тому що для них головне - це забезпечення нормальної життєдіяльності.

Як ви вже переконались, кількість моделей та їх різноманітність дуже велика. Щоб не загубитися в цьому розмаїтті, необхідно мати якусь класифікацію моделей. Розглянемо найбільш суттєві ознаки, за якими класифікуються моделі:

- галузі використання;
- урахування в моделі фактора часу;
- спосіб представлення моделей.

Розглядаючи моделі з позиції галузі використання, можна сказати, що вони бувають:

- учбові - наочні посібники, тренажери, навчальні програми;
- дослідні - створюються для дослідження характеристик реального об'єкта (модель теплоходу перевіряється на усталеність, а модель літака - на аеродинамічні характеристики);
- науково-технічні - для дослідження процесів та явищ (ядерний реактор або синхрофазотрон);
- ігрові моделі - для вивчення можливої поведінки об'єкта в запрограмованих або непередбачених ситуаціях (військові, економічні, спортивні ігри тощо);
- імітаційні моделі - виконується імітація дійсної ситуації, що багато повторюється для вивчення реальних обставин (випробування лікарських препаратів на мишах або інших тваринах, політ собаки в космос).

З урахуванням фактора часу моделі можуть бути динамічні та статичні. В першому випадку над об'єктом виконуються дослідження на протязі деякого терміну, а в другому - робиться одноразовий зріз стану (наприклад, постійний нагляд сімейного лікаря та одноразове обстеження в поліклініці).

За способом представлення моделі можуть бути матеріальні та інформаційні. *Матеріальні моделі* – це предметне відображення об'єкта зі збереженням геометричних та фізичних властивостей. Наприклад, іграшки, чучела тварин, географічні карти, глобус і таке інше - це матеріальні моделі реально існуючих об'єктів. Матеріальною моделлю можна також назвати хімічний або фізичний дослід. Ці моделі реалізують матеріальний підхід до вивчення об'єкта чи явища.

Інформаційна модель – це сукупність інформації, що характеризує властивості та стан об'єкта, процесу чи явища, а також взаємодію із зовнішнім світом. Інформаційні моделі можуть бути:

- вербальними - моделі отримані в результаті розумової діяльності людини і представлені в розумовій або словесній формі;

- знаковими - моделі, що виражені спеціальними знаками (малюнками, текстами, схемами, графіками, формулами тощо).

За формою представлення можна виділити наступні види інформаційних моделей:

- геометричні - графічні форми та об'ємні конструкції;
- словесні - усні та письмові описи з використанням ілюстрацій;
- математичні - математичні формули, що відображають зв'язок різних параметрів об'єкта;
- структурні - схеми, графіки, таблиці;
- логічні - моделі, в яких представлені різні варіанти вибору дій на основі різних умовиводів та аналізу умов;
- спеціальні - ноти, хімічні формули і таке інше;
- комп'ютерні та некомп'ютерні.

В сучасному світі розв'язування складних наукових та виробничих задач неможливе без використання моделей та моделювання. Серед різних видів моделей особливе місце займають математичні моделі, тому що вони дозволяють враховувати кількісні та просторові параметри явищ та використовувати точні математичні методи.

Вивчення реальних явищ за допомогою математичних моделей, як правило, вимагає застосування обчислювальних методів. При цьому широко використовуються методи прикладної математики, математичної статистики та інформатики. При вивченні даного курсу ви складатимете та працюватимете переважно з математичними комп’ютерними моделями.

Поняття алгоритму. Властивості алгоритму. Способи описування алгоритмів.

Кожна людина щодня зустрічається з безліччю задач від найпростіших і добре відомих до дуже складних. Для багатьох задач існують визначені правила (інструкції, команди), що пояснюють виконавцю, як розв'язувати дану проблему. Ці правила людина може вивчити чи заздалегідь сформулювати сама в процесі розв'язування задачі. Чим точніше описані правила, тим швидше людина опанує ними і буде ефективніше їх застосовувати.

У нашому житті ми постійно складаємо опис деякої послідовності дій для досягнення бажаного результату, тому поняття алгоритму не є для нас чимось новим і незвичайним. Так, ранком мама перед виходом до школи, дає вказівку: "Коли прийдеш зі школи, відразу пообідай і вимий посуд. Після цього підмети підлогу, сходи в магазин і можеш трохи погуляти. Гуляти дозволяю не більше години, а потім відразу за уроки".

Ця інструкція складається з послідовності окремих вказівок, що і визначають твою поведінку після повернення зі школи. Це і є алгоритм.

Кожний з нас використовує сотні різних алгоритмів. Спробуйте згадати деякі з них (алгоритми виконання арифметичних дій, розв'язування задач, прибирання квартири, миття посуду, готування їжі - рецепти тощо).

Отже, після обговорення кількох прикладів алгоритмів, давайте спробуємо сформулювати визначення, що ж таке алгоритм.

Алгоритмом називається зрозуміле і точне розпорядження виконавцю виконати послідовність дій, спрямованих на досягнення зазначеної мети чи на розв'язання поставленої задачі.

В цьому означенні використовується поняття "виконавець". Що це означає? Під *виконавцем алгоритму* ми розуміємо будь-яку істоту (живу чи неживу), яка спроможна виконати алгоритм. Все залежить від того, якої мети ми намагаємося досягнути. Наприклад: риття ями (виконавці - людина або екскаватор), покупка деяких товарів (один із членів родини), розв'язування математичної задачі (учень або комп'ютер) тощо.

Поняття алгоритму в інформатиці є фундаментальним, тобто таким, котре не визначається через інші ще більш прості поняття (для порівняння у фізиці – поняття простору і часу, у математиці – крапка).

Будь-який виконавець (і комп'ютер зокрема) може виконувати тільки обмежений набір операцій (екскаватор риє яму, вчитель вчить, комп'ютер виконує арифметичні дії). Тому алгоритми повинні мати наступні властивості:

1. *Зрозумілість*. Щоб виконавець міг досягти поставленої перед ним мети, використовуючи даний алгоритм, він повинен уміти виконувати кожну його вказівку, тобто розуміти кожну з команд, що входять до алгоритму. Наприклад: Мамі потрібно купити в магазині їжу. Виконавцем цього алгоритму може бути хтось із родини: батько, син, бабуся, маленька дочка тощо. Зрозуміло, що для тата достатньо сказати, які купити продукти, а далі деталізувати алгоритм не потрібно. Дорослому сину-підлітку необхідно детальніше пояснити в яких магазинах можна придбати потрібний товар, що можна купити замість відсутнього товару і таке інше. Маленькій дочці алгоритм необхідно деталізувати ще більше: де взяти сумку, щоб принести товар, яку решту грошей необхідно повернути з магазину, як дійти до магазину і як там поводитись (якщо дитина вперше йде за покупками).

2. *Визначеність (однозначність)*. Зрозумілий алгоритм все ж таки не повинен містити вказівки, зміст яких може сприйматися неоднозначно. Наприклад, вказівки "почисти картоплю", "посоли за смаком", "прибери в квартирі" є неоднозначними, тому що в різних випадках можуть призвести до різних результатів. Крім того, в алгоритмах неприпустимі такі ситуації, коли після виконання чергового розпорядження алгоритму виконавцю не зрозуміло, що потрібно робити на наступному кроці. Наприклад: вас послали за яким-небудь товаром у магазин, та ще попередили "без (хліба, цукру і таке інше) не повертайся", а що робити, якщо товар відсутній? Отож, точність – це властивість алгоритму, що полягає в тім, що алгоритм повинен бути однозначно витлумачений і на кожному кроці виконавець повинен знати, що йому робити далі.

3. *Дискретність*. Як було згадано вище, алгоритм задає повну послідовність дій, які необхідно виконувати для розв'язання задачі. При цьому, для виконання цих дій їх розбивають у визначеній послідовності на прості кроки. Виконати дії наступного розпорядження можна лише виконавши дії попереднього. Ця розбивка алгоритму на окремі елементарні дії (команди), що легко виконуються даним виконавцем, і називається дискретністю.

4. *Масовість*. Дуже важливо, щоб складений алгоритм забезпечував розв'язання не однієї окремої задачі, а міг виконувати розв'язання широкого класу задач даного типу. Наприклад, алгоритм покупки якого-небудь товару буде завжди однаковий, незалежно від товару, що купується. Або алгоритм прання не залежить від білизни, що переться, і таке інше. Отож, під масовістю алгоритму мається на увазі можливість його застосування для вирішення великої кількості однотипних завдань.

5. *Результативність*. Взагалі кажучи, очевидно, що виконання будь-якого алгоритму повинне завершуватися одержанням кінцевих результатів. Тобто ситуації, що в деяких випадках можуть призвести до так званого "зациклення", повинні бути виключені при написанні алгоритму. Наприклад, розглянемо таку ситуацію: роботу дано завдання залишити кімнату (замкнутий простір), не виконуючи руйнівних дій. У цьому випадку, якщо роботу не дати вказівки відкрити двері (що, можливо, закриті), то спроби залишити приміщення можуть бути безуспішними.

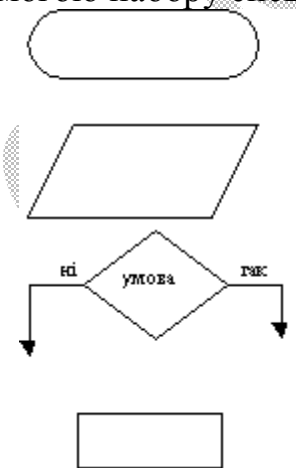
Базові структури алгоритмів.

Тепер залишається з'ясувати, яким чином можна подати алгоритм виконавцю. Існує кілька методів запису алгоритмів, вибір яких залежить від виконавця та того, хто його задає.

Перший спосіб – це *словесний опис алгоритму*.

Другий спосіб - це подача алгоритму *у вигляді таблиць, формул, схем, малюнків* тощо. Наприклад, всіх вас вчили в дитячому садочку, в школі, в родині правилам поведінки на дорозі. І найкраще сприймається алгоритм, що поданий у вигляді схематичних малюнків. Дивлячись на них, людина відпрацьовує ту лінію поведінки, що їй пропонується.

Третій спосіб - запис алгоритмів за допомогою *блок-схеми*. Цей метод був запропонований в інформатиці для наочності представлення алгоритму за допомогою набору спеціальних блоків. Основні з цих блоків наступні:



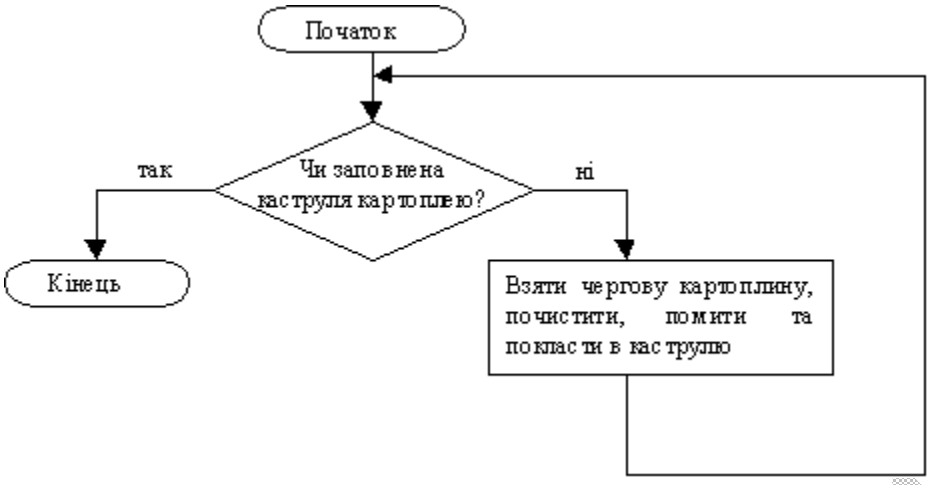
початок або кінець алгоритму

блок введення-виведення даних

блок умовного переходу (розгалуження)

командний блок

Використовуючи дані блоки, можна подати, наприклад, алгоритм чищення картоплі в такому вигляді:



П'ятий спосіб максимально наближений до комп'ютера - це *мови програмування*. Справа в тому, що найчастіше в практиці виконавцем створеного людиною алгоритму являється машина і тому він повинен бути написаний мовою, зрозумілою для комп'ютера, тобто мовою програмування.

При вивченні даного курсу ми будемо складати алгоритми розв'язування математичних задач та подавати їх у вигляді блок-схем та програм мовою програмування Турбо Паскаль (Turbo Pascal).

Базові структури алгоритмів

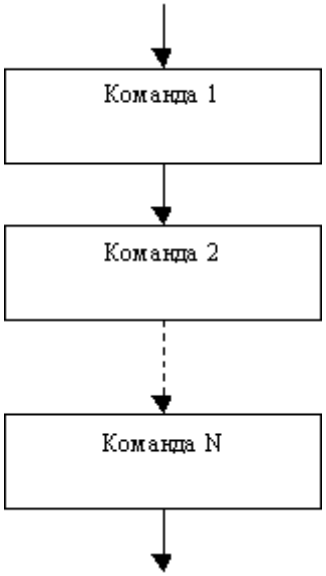
Навіть ще не маючи досвіду в створенні алгоритмів, ми інтуїтивно розуміємо, що вони розрізняються за своєю структурою. Так є алгоритми, що виконуються за будь-яких обставин. Але таке трапляється нечасто, тому що людина завжди коригує свої плани в залежності від оточуючих умов і тому виникає ситуація "якщо трапиться...", "якщо зустрінуся...", "якщо встигну..." тощо. А іноді ми змушені повторювати якийсь процес кілька разів, доки не отримаємо бажаного результату. Найчастіше ж ми і умови враховуємо, і повторюємо щось. Ось так і виникають різні типи алгоритмів.

Всього існують чотири базових структури алгоритмів:

- лінійні;
- розгалужені;
- циклічні;
- змішані.

Найпростіша в написанні та виконанні перша з цих структур – лінійна. До неї відносяться алгоритми, що складаються лише з простих команд. Які ж команди можна назвати простими? Простими з точки зору комп'ютера являються ті команди, що виконуються виконавцем безумовно, тобто після першої команди виконується друга, потім третя і т.д.

Загальний вигляд лінійного алгоритму, поданий мовою блок-схем, наступний:



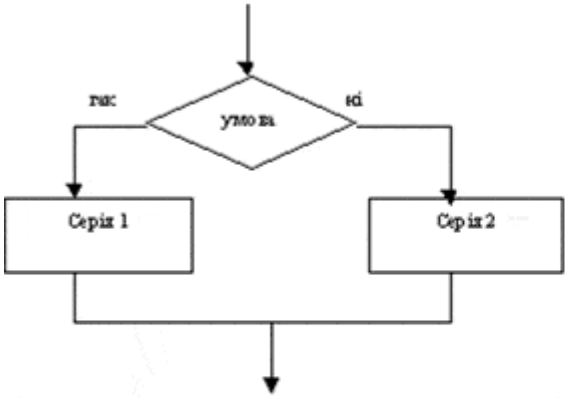
На відміну від людини виконавець "комп'ютер" не може відмовитися від виконання команди, він не може подібно недбалому учню сказати "не хочу", "не можу", " в мене болить голова і поганий настрій". Команда, записана в алгоритмі, повинна бути виконаною, тому, якщо знехтувати суто людськими якостями ("не хочу", "не можу" і т.д.), лінійним можна назвати алгоритм ранкового збирання до школи або до університету.

1) проснутися;
2) зробити ранковий туалет;
3) одягнутися;
4) поснідати;
5) зібрати речі;
6) одягнути верхній одяг;
7) вийти до школи (університету).

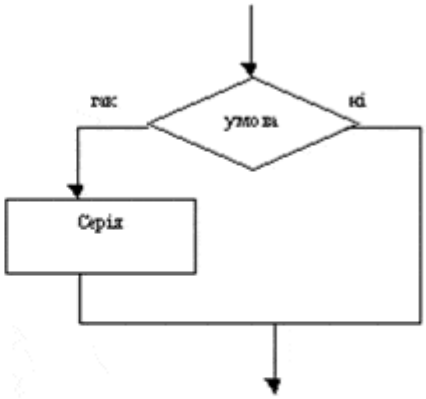
Та, навіть, в такому простому алгоритмі в зразу ж знайдете недоліки. А що робити, якщо я себе погано почуваю (захворів), а якщо я вже зібрав речі ввечері, а якщо я не встиг напередодні вивчити всі уроки і мені необхідно щось повторити, а що значить одягнути верхній одяг (він залежить від пори року, погоди тощо). Якщо ж спробувати прослідкувати за вашою поведінкою на протязі дня, то з'ясується, що майже ніколи ви не дієте за лінійним алгоритмом. Весь час ви аналізуєте ситуацію, змінюєте свою поведінку та свої плани, пристосовуєтесь до обставин.

Тому набагато частіше зустрічається другий тип алгоритму – *розгалужений*. Цей алгоритм обов'язково містить в собі хоча б одну умову (як правило, їх набагато більше) і виконується він в залежності від цієї умови.

Мовою блок-схем розгалужений алгоритм подається наступним чином:



Повна форма команди розгалуження



Неповна форма команди розгалуження

Тепер розберемось, що ж таке умова з точки зору виконавця. Умовою називається таке речення, на яке можна дати відповідь "так" чи "ні". Як правило, кажуть, що в першому випадку (коли ми відповіли на речення "так") умова являється істиною, а в другому хибною.

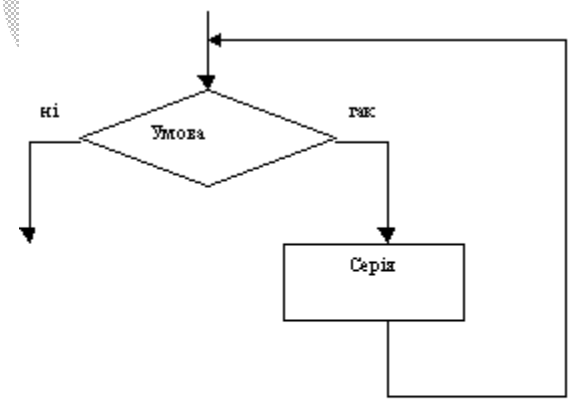
У нашому житті дуже часто зустрічаються алгоритми з повторами, причому чітко визначаються два типи повторів. В одному випадку ми чітко знаємо, скільки разів необхідно повторити задану послідовність команд, а в іншому – ні.

В залежності від того, чи знаємо ми скільки разів необхідно повторювати якусь послідовність команд розрізняють цикли з параметром (кількість повторень відома заздалегідь) та цикли з умовою (цикл робиться доки не виконається якась умова).

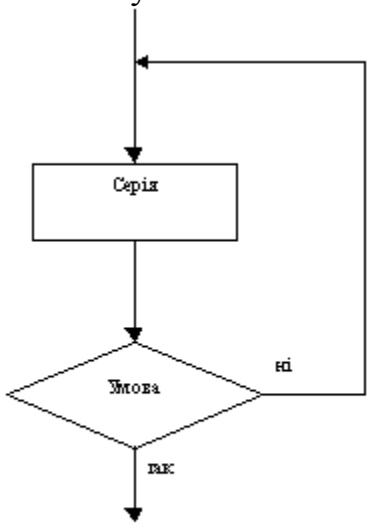
Крім того, в циклах з умовою теж можна виділити два різних випадки:

- цикл з передумовою - коли ми спочатку перевіряємо умову, а потім виконуємо деяку послідовність дій;
- цикл с післяумовою - спочатку ми виконуємо хоч один раз необхідну послідовність дій, а потім перевіряємо, чи не досягли ми бажаного результату.

Мовою блок-схем обидва типи циклів виглядають наступним чином:



Цикл з передумовою



Цикл с післяумовою

Найчастіше ж у житті ми використовуємо змішані алгоритми, в яких поєднуються елементи лінійних, розгалужених та циклічних алгоритмів.

Поняття програми. Класифікація мов програмування.

Процес роботи комп'ютера полягає у виконанні програм, тобто деякого набору команд, що надходять у визначеному порядку. Машинний код команди складається з нулів та одиниць та указує, яку саме дію треба виконати центральному процесору. Отже, щоб задати комп'ютеру послідовність дій, яку він має виконати, треба задати послідовність двійкових кодів відповідних команд. Писати такі програми дуже складна справа. Раніше для цього програміст повинен був пам'ятати не тільки всі комбінації нулів та одиниць двійкового коду кожної команди, але й двійкові коди адрес даних, що використовувалися під час виконання програми. Щоб полегшити роботу програмістів, було розроблено багато мов програмування, які в більш наочному для людини вигляді подавали послідовність дій комп'ютера.

Алгоритмічні мови, призначені для побудови описів алгоритмів, що виконуються електронними обчислювальними машинами, називаються *мовами програмування*.

Описи алгоритмів мовою програмування називають *програмами*.

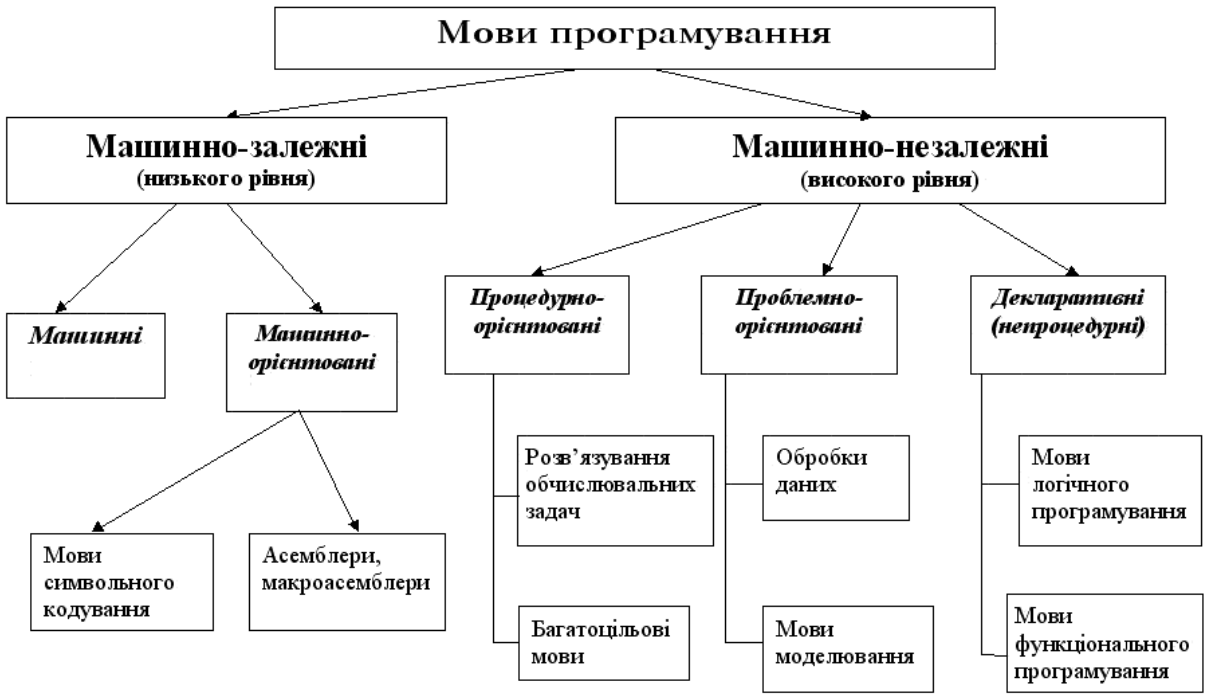
Набагато легше написати програму мовою, що наближена до людської, а перекладання цієї програми на машинні коди доручити комп'ютеру. Так з'явилися мови, що призначені спеціально для написання програм – мови програмування.

Існує багато різних мов програмування (дивись малюнок). Взагалі, для розв'язування більшості задач можна використовувати будь-яку з них. Тільки досвідчені програмісти знають, яку мову програмування краще використовувати для розв'язування складних спеціалізованих задач, щоб урахувати особливості тієї чи іншої з них.

Всі існуючи мови програмування можна поділити на дві групи:

- мови низького рівня;
- мови високого рівня.

До мов низького рівня належать мови асемблера (від англ. to assemble - складати, компонувати). У мові асемблера використовуються символьні позначення команд, які легко зрозуміти і запам'ятати. Замість послідовностей двійкових кодів команд записуються їх символьні позначення, а замість двійкових адрес даних, які використовуються під час виконання програми, - символьні імена цих даних. Іноді мову асемблера називають мнемокодом або автокодом.



Більшість програмістів при складанні програм користуються деякою мовою високого рівня. Для описування алгоритмів такою мовою використовується певний набір символів – алфавіт мови. З цих символів складаються так звані службові слова мови, кожне з яких має певне призначення. Службові слова зв'язуються одне з одним в речення за певними синтаксичними правилами мови і визначають деяку послідовність дій, які мусить виконати комп'ютер.

Використання мов високого рівня надає можливість описувати програми для комп'ютера, використовуючи загальноприйняті позначення операцій і функцій.

Та програми, що написані на мовах програмування високого рівня (алгоритмічних мовах програмування), комп'ютер "не розуміє". Для того, щоб він міг виконати програму, її потрібно перекласти на машинну мову. Для такого перекладу використовують спеціальні програми, що мають назву – *транслятори*.

Транслятор – це програма, що призначена для перекладу тексту програми з однієї мови програмування на іншу. Процес перекладання називається *трансляцією*.

Розрізняють два типи трансляторів:

- компілятори
- інтерпретатори.

Компілятор – це програма, призначена для перекладу в машинні коди програми, що написана мовою високого рівня. Процес такого перекладання називається *компіляцією*.

Кінцевим результатом роботи компілятора є програма в машинних кодах, яка потім виконується ЕОМ. Скомпільований варіант програми можна зберігати на дискові. Для повторного виконання програми компілятор вже не потрібен. Досить завантажити з диска в пам'ять комп'ютера скомпільований перед цим варіант і виконати його.

Існує інший спосіб поєднання процесів трансляції та виконання програм. Він називається інтерпретацією.

Інтерпретатор – це програма, що призначена для трансляції та виконання вихідної програми по командах (на відміну від транслятора, який цей процес виконує в цілому). Такий процес називається *інтерпретацією*.

У процесі трансляції відбувається перевірка програми на відповідність до правил її написання. Якщо в програмі знайдені помилки, транслятор виводить повідомлення про них на екран монітора. Інтерпретатор повідомляє про знайдені помилки після трансляції кожної команди програми, а компілятор - після завершення компіляції всієї програми. Знайти та виправити в цьому випадку помилки значно складніше, ніж при інтерпретації. Через це програми-інтерпретатори розраховані, в основному, на мови, що призначені для навчання програмуванню, і використовуються програмістами-початківцями.

Як правило, програми компілятори та інтерпретатори називаються так само, як і мови, для перекладу з яких вони призначені. Слова Паскаль, Бейсік, Сі можна сприймати і як назви мов, і як назви відповідних програм-трансляторів.

Однією з найпопулярніших мов програмування є мова Паскаль, яку створив в 1968 році швейцарський вчений Ніклаус Вірт. Яку, доречи, ми й будемо вивчати в даному курсі.

ІСТОРІЯ РОЗВИТКУ МОВ ПРОГРАМУВАННЯ

Загальновідомо, що інформаційні технології є однією з областей життя, що найшвидше розвивається. Нові технології, назви та абривіатури, проекти з’являються ледве не кожен день.

Перші універсальні мови

Звернемося до витоків розвитку обчислювальної техніки. Згадаємо найперші комп’ютери та програми для них. Це була ера програмування безпосередньо в машинних кодах, а основним носієм інформації були перфокарти та перфоленти. Програмісти повинні були знати архітектуру машини досконально. Програми були досить проситми, що було обумовлено, по-перше, всіма обмеженими можливостями цих машин, та, по-друге, великою складністю розробки та, головне, налагодження програм безпосередньо на машинних мовах. Разом з тим такий спосіб розробки давав програмісту просте величезну владу над системою. З’являлася можливість таких хитрих алгоритмів та способів організації програм, які й не снилися сучасним розробниками. Наприклад, використовувалась така можливість, як код, що

самомодифікувався. Знання двійкового представлення команд дозволяло іноді не зберігати деякі дані окремо, а вбудовувати їх у код як оманди. І це далеко не повний перелік прийомів, володіння хоча б одним з яких зараз зразу ж просуває вас до рівня «гуру» екстра-класу.

Ассемблер

Першим значним кроком став перехід до мови ассемблера (англійська мова assembly language, або assembler, на українську перекладається саме тим терміном, який було використано вище). Не дуже помітний, з першого поляду крок – перехід до символьного кодування машинних команд – мав насправді величезне значення. Програмісту не треба було більше занурюватися до способів кодування команд на апаратному рівні. Більш того, зазвичай однакові за суттю команди кодувались зовсім різними способами залежно від власних параметрів. З'явилася також можливість використання макросів та міток, що також спрощувало створення, модифікацію та налагодження програм. З'явилася навіть деяка подібність переносимості – існувала можилвість розробки цілого сімейства машин зі схожою системою команд та деякого загального ассемблера для них, при цьому не було потреби забезпечувати двійкову сумісність. Разом з тим, перехід до нової мови мав і деякий негативний бік. Ставало майже неможливим використання усяких хитрих прийомів, деякі з яких були вже згадані. Крім того, тут вперше в історії розвитку програмування з'явилося два представлення програми: у вихідних текстах та у відкомпільованому вигляді. До кінця ассемблерної ери можливість автоматичної трансляції в обидва боки була втрачена назавжди.

Фортран

В 1954 році у надрах IBM групою розробників на чолі з Джоном Бекусом (John Backus) було створено мову програмування Fortran. Значення цієї події важко переоцінити. Це перший крок програмування високого рівня. Вперше програміст міг посправжньому абстрагуватися від

особливостей машинної архітектури. Ключовою ідеєю, що відрізняв нову мову від ассемблера, була концепція підпрограм. Крім того, синтаксична структура мови була достатньо складна для машинної обробки в першу чергу через те, що пробіли як синтаксичні одиниці взагалі не використовувались. Це породжувало масу помилок. Мова Фортран використовувалася (і використовується на сьогоднішній день) для наукових обчислень. Він страждає від відстутності багатьох звичних мовних конструкцій та атрибутів, компілятор практично ніяк не перевіряє синтаксично правильну програму з точки зору семантичної коректності. В ньому немає підтримки сучасних способів структурування коду та даних. Це усвідомлювали і самі розробники. За визнанням самого Бекуса, перед ними стояла задача скоріше розробки компілятора, аніж мови. Розуміння самостійного значення мов програмування прийшло пізніше.

Появу Фортрана зустріли ще з більшою критикою, аніж ассемблера. Програмістів лякало зниження ефективності програм за рахунок використання проміжної ланки у виді компілятора. І ці побоювання мали під собою основу: дійсно, гарний програміст, скоріш за все, при розв’язуванні якої-небудь невеликої задачі вручну напише код, що працюватиме швидше, аніж код, отриманий як результат компіляції. Через деякийчас прийшло розуміння того, що реалізація великих проектів неможлива без використання мов високого рівня. Потужність обчислювальних машин зростала, і переваги мов високого рівня стали настільки очевидними, що підштовхнуло розробників до творення нових мов, все більш сучасних.

Cobol

В 1960 році була створена мова програмування Cobol. Вона задумувалася як мова для створення комерційних додатків, і він став таким. На Коболі написані тисячі прикладних комерційних систем. Відмітною особливістю мови є можливість ефективної роботи з великими масивами даних, що характерне саме комерційним додаткам. Популярність Кобола настільки висока, що навіть зараз, при всіх його недоліках (за структурою та задуму Кобол нагадує Фортран) з’являються нові йогодіалекти та реалізації. Так нещодавно з’явилась

реалізація Кобола, у поєднанні з Microsoft.NET, що вимагало внесення до мови деяких рис об'єктно-орієнтованої мови.

PL/1

В 1964 році все таж сама корпорація IBM створила мову PL/1, яка була покликана замінити Кобол та Фортран у більшості додатках. Мова володіла винятковим багатством синтаксичних конструкцій. В ній вперше з'явилася обробка виняткових ситуацій та підтримка паралелізму. Треба відмітити, що синтаксична структура мови була украй складною. Пропуски вже використовувалися як синтаксичні роздільники, але ключові слова не були зарезервовані. Зокрема, наступний рядок - це цілком нормальний оператор на PL/1:

IF ELSE=THEN THEN THEN; ELSE ELSE

В силу таких особливостей розробка компілятора для PL/1 була виключно складною справою. Мова так і не стала популярною поза світом IBM.

BASIC

В 1963 році Дартмурським коледжом (Dartmouth College) була створена мова програмування BASIC (Beginners' All-Purpose Instruction Code – багатоцільова мова символічних інструкцій для початківців). Мова замислювалася впершу чергу як засіб навчання і як перша мова програмування, що вивчалася. Він передбачався таким, що легко інтерпретується і компілюється. Треба сказати, що BASIC дійсно став мовою, на якою вчаться програмувати. Було створено декілька потужних реалізацій BASIC, що підтримують найсучасніші концепції програмування (яскравий приклад - Microsoft Visual Basic).

Algol

У 1960 році командою на чолі з Петером Науром (Peter Naur) була створена мова програмування Algol. Ця мова дала початок цілому сімейству

Алгол-подібних мов (найважливіший представник - Pascal). У 1968 році з'явилася нова версія мови. Вона не знайшла такого широкого практичного застосування, як перша версія, але була вельми популярна в кругах теоретиків. Мова була достатньо цікава, оскільки володіла багатьма унікальним на той момент характеристиками.

Подальший розвиток мов програмування

Створення кожна з вищезазначених мов (за виключенням, можливо, Algol'a) було викликано деякими практичними вимогами. Ці мови послужили фундаментом для пізніших розробок. Всі вони представляють одну і ту ж парадигму програмування. Наступні мови пішли істотно далі в своєму розвитку, убік глибшого абстрагування.

Відомості про пізніші мови я наводитиму у вигляді опису сімейств мов. Це дозволить краще прослідкувати взаємозв'язки між окремими мовами

Pascal-подібні мови

У 1970 році Никлаусом Віртом було створено мову програмування Pascal. Мова чудова тим, що це перша широко поширена мова для структурного програмування (першим, строго кажучи, був Алгол, але він не набув такого широкого поширення). Вперше оператор безумовного переходу перестав грати основоположну роль при управлінні порядком виконання операторів. У цій мові також упроваджена строга перевірка типів, що дозволило виявляти багато помилок на етапі компіляції.

Негативною рисою мови була відсутність в ній засобів для розбиття програми на модулі. Вірт усвідомлював це і розробив мову Modula-2 (1978), в якій ідея модуля стала однією з ключових концепцій мови. У 1988 році з'явилася Modula-3, в яку були додані об'єктно-орієнтовані риси. Логічним продовженням Pascal і Modula є мова Oberon і Oberon-2. Вони характеризуються рухом убік об'єктно- і компонентно- орієнтованості.

С-подібін мови

У 1972 році Керніганом і Рітчи була створена мова програмування С. Вона створювалася як мова для розробки операційної системи UNIX. *С* часто називають «асемблером, що переноситься», маючи на увазі те, що він дозволяє працювати з даними практично так само ефективно, як на асемблері, надаючи при цьому структуровані конструкції, що управляють, і абстракції високого рівня (структури і масиви). Саме з цим пов'язана його величезна популярність і понині. І саме це є його ахіллесовою п'ятою. Компілятор *С* дуже слабо контролює типи, тому дуже легко написати зовні абсолютно правильну, але логічно помилкову програму.

У 1986 році Бьярн Страуструп створив першу версію мови С++, додавши в мову С об'єктно-орієнтовані риси, узяті з Simula, і виправивши деякі помилки і невдалі рішення мови. С++ продовжує удосконалюватися і в даний час, так в 1998 році вийшла нова (третя) версія стандарту, що містить в собі деякі досить істотні зміни. Мова стала основою для розробки сучасних великих і складних проектів. У ній є, проте і слабкі сторони, що витікають з вимог ефективності.

У 1995 році в корпорації Sun Microsystems Кеном Арнольдом і Джеймсом Гослінгом була створена мова Java. Він успадковував синтаксис С і С++ і був позбавлений від деяких неприємних рис останнього. Відмітною особливістю мови є компіляція в код якоїсь абстрактної машини, для якої потім пишеться емулятор (Java Virtual Machine) для реальних систем. Крім того, в Java немає показників і множинного спадкоємства, що сильно підвищує надійність програмування.

У 1999-2000 роках в корпорації Microsoft була створена мова C#. Вона достатньою мірою схожа з Java (і замислювалася як альтернатива останній), але має і відмітні особливості. Орієнтована, в основному, на розробку багатокомпонентних Інтернет-додатків.

Мови Ada та Ada 95

У 1983 році під егідою Міністерства Оборони США була створена мова Ada. Мова чудова тим, що дуже багато помилок може бути виявлено на етапі компіляції. Крім того, підтримуються багато аспектів програмування, які часто

віддаються на відкуп операційній системі (паралелізм, обробка виключень). У 1995 році був прийнятий стандарт мови Ada 95, яка розвиває попередню версію, додаючи в неї об'екну орієнтованість і виправляючи деякі неточності. Обидві ці мови не отримали широкого розповсюдження поза військовими й іншими великомасштабними проектами (авіація, залізничні перевезення). Основною причиною є складність освоєння мови і достатньо громіздкий синтаксис (значно громіздкіший, ніж Pascal).

Мови обробки даних

Всі вищеперелічені мови є мовами загального призначення в тому сенсі, що вони не орієнтовані і не оптимізовані під використання яких-небудь специфічних структур даних або на застосування в яких-небудь специфічних областях. Було розроблено велику кількість мов, орієнтованих на достатньо специфічні застосування. Розглянемо діякі з них.

APL

У 1957 році була зроблена спроба створення мови для опису математичної обробки даних. Мова була названа APL (Application Programming Language). Його відмітною особливістю було використання математичних символів (що ускладнювало застосування на текстових терміналах; поява графічних інтерфейсів зняла цю проблему) і дуже могутній синтаксис, який дозволяв проводити безліч нетривіальних операцій прямо над складними об'єктами, не вдаючись до розбиття їх на компоненти. Широкому застосуванню перешкодило, як вже наголошувалося, використання нестандартних символів як елементів синтаксису.

Snobol та Icon

У 1962 році з'явилася мова Snobol (а в 1974 - його наступник Icon), призначена для обробки рядків. Синтаксис Icon нагадує C і Pascal одночасно. Відмінність полягає в наявності потужних вбудованих функцій роботи з рядками і пов'язана з цими функціями особлива семантика. Сучасним аналогом

Icon і Snobol є Perl - мова обробки рядків і текстів, в яку додано деякі об'єктно-орієнтовані можливості. Вважається дуже практичною мовою, проте їй бракує елегантність.

SETL

У 1969 році була створена мова SETL - мова для опису операцій над множинами. Основною структурою даних в мові є множина, а операції аналогічні математичним операціям над множинами. Корисна при написанні програм, що мають справу з складними абстрактними об'єктами.

Скриптові мови

Останнім часом у зв'язку з розвитком Інтернет-технологій, широким розповсюдженням високопродуктивних комп'ютерів і низки інших чинників набули поширення так звані скриптові мови. Ці мови спочатку орієнтувалися на використання в якості внутрішніх керуючих мов у складних системах. Багато хто з них, проте ж, вийшов за межі сфери свого початкового застосування і використовуються нині в зовсім інших областях. Характерними особливостями даних мов є, по-перше, їх інтерпретованість (компіляція або неможлива, або небажана), по-друге, простотий синтаксис, а по-третє, легка розширюваність. Таким чином, вони ідеально підходять для використання в часто змінних програмах, дуже невеликих програмах або у випадках, коли для виконання операторів мови витрачається час, нечпівставний з часом їх розбору. Було створено чималу кількість таких мов, перерахуємо лише основні.

JavaScript

Мова була створена в компанії Netscape Communications як мова для опису складної поведінки веб-сторінок. Спочатку називалася LiveScript, причиною зміни назви стали маркетингові міркування. Інтерпретується браузером під час відображення веб-сторінки. За синтаксисом схожа з Java і (віддалено) з C/C++. Має можливість використовувати вбудовану в браузер об'єктну функціональність, проте об'єктно-орієнтованою мовою не є.

VBScript

Мова була створена в корпорації Microsoft багато в чому як альтернатива JavaScript. Має схожу область застосування. Синтаксично схожий з мовою Visual Basic (і є урізаною версією останньої). Так само, як і JavaScript, виконується браузером при відображенні веб-сторінок і має той ступінь об'єктно- орієнтованості.

Perl

Мова створювалася в допомогу системному адміністраторові операційної системи Unix для обробки різного роду текстів і виділення потрібної інформації. Розвинулася до потужного засобу роботи з текстом. Є мовою, що інтерпретується, і реалізована практично на всіх існуючих платформах. Застосовується при обробці текстів, а також для динамічної генерації веб-сторінок на веб-серверах.

Python

Об'єктно-орієнтована мова програмування, що інтерпретується. За структурою та областю застосування близька до Perl, проте менш поширена та строгіша й логічніша. Є реалізації для більшості існуючих платформ.

Об’єктно-орієнтовані мови

Об'єктно-орієнтований підхід, що прийшов на зміну структурному, вперше з'явився зовсім не в C++, як вважають деякі. Існує ціла низка чистих об'єктно-орієнтованих мов, без відомостей про які наш огляд був би неповним.

Simula

Першою об'єктно-орієнтрованою мовою була мова Simula (1967). Ця мова була призначена для моделювання різних об'єктів і процесів, і об'єктно-орієнтовані риси з'явилися в ній саме для опису властивостей модельних об'єктів.

Smalltalk

Популярність об'єктно-орієнтованому програмуванню принесла мова Smalltalk, створену в 1972 році. Мова призначалася для проектування складних графічних інтерфейсів і була першою по-справжньому об'єктно-орієнтованою мовою. У ній класи і об'єкти - це єдині конструкції програмування. Великим недоліком Smalltalk є великі вимоги до пам'яті і низька продуктивність отриманих програм. Це пов'язано з не дуже вдалою реалізацією об'єктно-орієнтованих особливостей.

Eiffel

Існує мова з дуже вдалою реалізацією об'єктної-орієнтованості, що не є надбудовою ні над якою іншою мовою. Це мова Eiffel (1986). Будучи чистою мовою об'єктно-орієнтованого програмування, вона, крім того, підвищує надійність програми шляхом використання «контрольних тверджень».

Мови паралельного програмування

Більшість комп'ютерної архітектури і мов програмування орієнтовані на послідовне виконання операторів програми. В даний час, проте, існують програмно-апаратні комплекси, що дозволяють організувати паралельне виконання різних частин одного і того ж обчислювального процесу. Для програмування таких систем необхідна спеціальна підтримка з боку засобів програмування, зокрема, мов програмування. Деякі мови загального призначення містять в собі елементи підтримки паралелізму, проте ж програмування суто паралельних систем потребує спеціальних прийомів. Однією з таких мов є Occam, Linda.

Неімперативні мови

Всі мови, про які йшлося раніше, мають одну загальну властивість: вони імперативні. Це означає, що програми на них, зрештою, є покроковим описом рішення тієї або іншої задачі. Можна спробувати описувати лише постановку

проблеми, а вирішувати задачу доручити компілятору. Існує два основні підходи, що розвивають цю ідею: функціональне і логічне програмування.

Функціональні мови

Основна ідея, що лежить в основі функціонального програмування, - це представлення програми у вигляді математичних функцій (тобто функцій, значення яких визначається лише їх аргументами, а не контекстом виконання). Оператор присвоєння в таких мовах не використовується (або, як мінімум, його використання не заохочується). Імперативні можливості, як правило, є, але їх застосування обставлене серйозними обмеженнями. Існують мови з пасивною та з активною семантикою. Відмінність полягає, грубо кажучи, в тому, що в мовах з активною семантикою обчислення проводяться в тому ж місці, де вони описані, а у разі пасивної семантики обчислення проводиться тільки тоді, коли воно дійсно необхідне. Перші мови мають ефективнішу реалізацію, тоді як другі - кращу семантику.

З мов з активною семантикою згадаємо ML і два його сучасних діалекту - Standard ML (SML) і CAML. Останній має об'єктно-орієнтованого нащадка - Objective CAML (O'CaML).

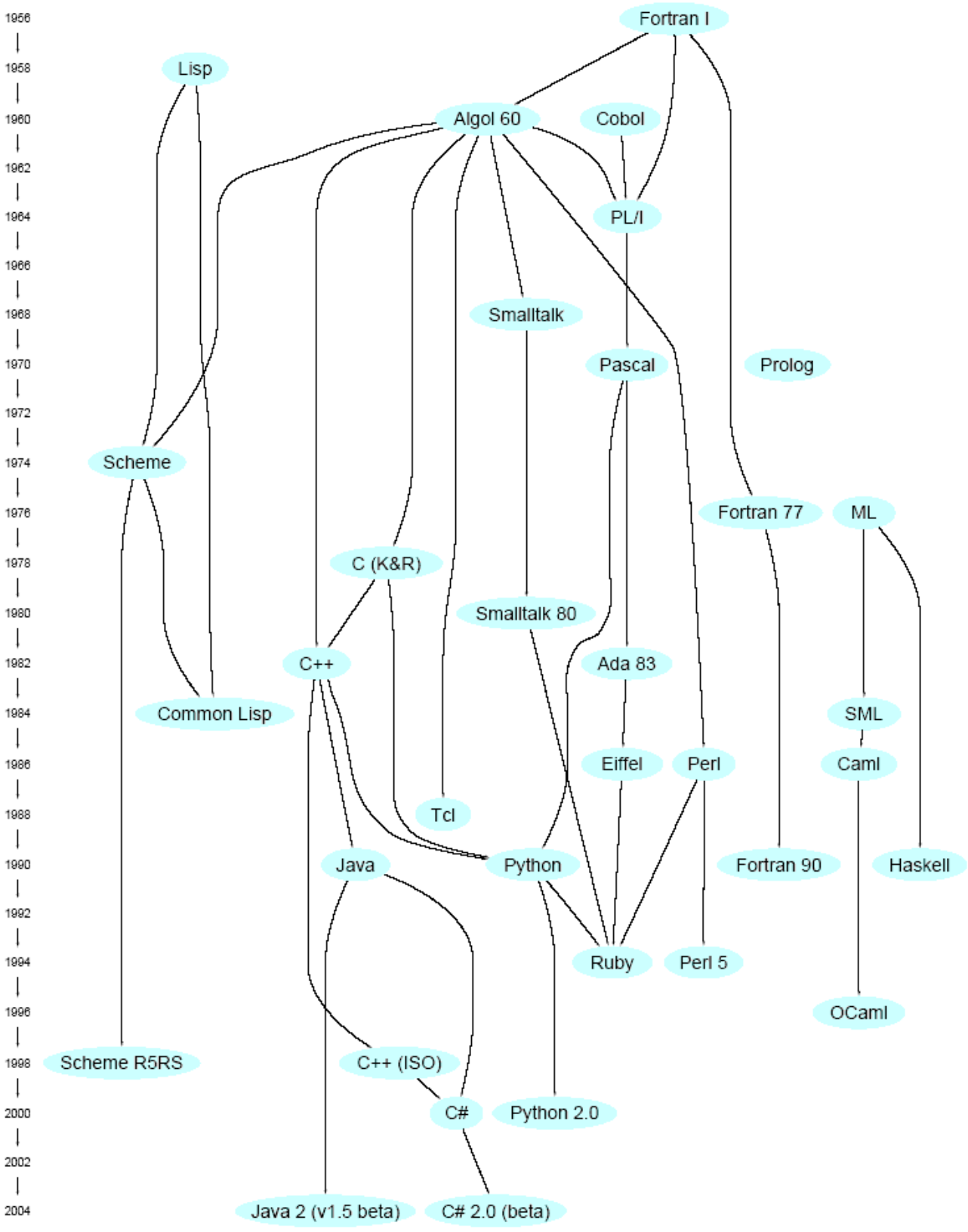
Серед мов з пасивною семантикою найбільш поширені два: Haskell і його більш простий діалект Clean.

Мови логічного програмування

Програми на мовах логічного програмування виражені як формули математичної логіки, а компілятор намагається отримати наслідки з них.

Родоначальником більшості мов логічного програмування є мова Prolog (1971). У ній є низка нащадків - Parlog (1983, орієнтований на паралельні обчислення), Delta Prolog та ін. Логічне програмування, як і функціональне, - це окрема область програмування, що розглядатися нами не буде.

Представимо на малюнку дерево розвитку мов програмування:



Мова Turbo Pascal

Зупинимося на мові, що нами буде вивчатися.

Паскаль (англ. Pascal) - мова програмування загального призначення.

Була створений Никлаусом Віртом в 1970, після його участі в роботі комітету розробки стандарту мови Алгол, як мова для навчання процедурному програмуванню. Назва мові дана на честь видатного французького математика, фізика, літератора і філософа Блеза Паскаля. Спочатку мова компілювалася в байт-код, подібно до мови Java.

Особливостями мови є строга типізація і наявність засобів структурного (процедурного) програмування. Паскаль був одним з перших таких мов. На думку Н. Вірта, мова повинна сприяти дисциплінуванню програмування, тому, разом із строгою типізацією, в Паскалі зведені до мінімуму можливі синтаксичні неоднозначності, а сам синтаксис інтуїтивно зрозумілий навіть при першому знайомстві з мовою.

Проте, спочатку мова володіла безліччю недоліків: неможливість передачі функціям масивів змінної довжини, відсутність нормальних засобів роботи з динамічною пам'яттю, обмежена бібліотека введення-виведення, відсутність засобів для підключення функцій написаних на інших мовах, відсутність засобів роздільної компіляції і т.п. Повний розбір недоліків мови Паскаль був виконаний Брайаном Керніганом в статті «Чому Паскаль не є моєю улюбленою мовою програмування». Необхідно відмітити, що багато перерахованих недоліків мови не виявляються або навіть стають перевагами при навчанні програмуванню. Крім того, основною мовою програмування в академічному середовищі 70-х був Фортран, що мав набагато істотніші недоліки, і Паскаль став значним кроком вперед.

Автор мови розумів недоліки створеної ним мови, перестав його розвивати і розробив мови Модула-2 і Оберон.

Проте, переваги мови примушували багато комерційних і некомерційних організацій розробляти системи програмування на основі мови Паскаль.

З числа останніх виділяється фірма Borland, Turbo Pascal (потім Borland Pascal) якою був значно розширений, було усунено багато недоліків мови, додані нові можливості. Мова стала багатшою, але одночасно, втратила переносимість та спільність.

Важливим кроком в розвитку мови, стала поява вільної мови Паскаль GNU Pascal, яка не тільки увібрала в себе риси інших Паскалів, не тільки дозволила нарешті повністю відмовитися від «брудних» прийомів програмування, особливо властивих, скажімо, Turbo Pascal, але і забезпечив широку портабельность написаних на нею програм (більше 20 різних платформ, під більш ніж 10 різними операційними системами).

Зараз користуються популярністю такі версії мови як TMT Pascal, Free Pascal і GNU Pascal. Продовжує використовуватися і Borland Pascal. Розвитком мови Borland Pascal є Object Pascal - версія мови Паскаль розширена засобами об'єктно-орієнтованого програмування. Останні версії Borland Pascal лежать в основі середовища програмування Delphi.