Змістовий модуль 1 Тема 1. ***Загальне поняття алгоритму. Алгоритмічні мови. Послідовність рішення задачі по розробці програми. Формалізація задачі, побудова математичної та інформаційної моделі. Розробка алгоритму рішення задачі. Складання програми. Тестування та відладка програми***

Загальне поняття алгоритму

У старому трактуванні алгоритм — це точний набір інструкцій, що описують послідовність дій виконавця для досягнення результату рішення задачі за кінцевий час. У міру розвитку паралельності в роботі комп'ютерів слово «послідовність» почали замінювати більш загальним словом «порядок». Це пов'язано з тим, що якісь дії алгоритму повинні бути виконані тільки один за одним, але якісь можуть бути і незалежними.

Часто як виконавець виступає деякий механізм (комп'ютер, токарний верстат, швейна машина), але поняття алгоритму необов'язково відноситься до комп'ютерних програм, так, наприклад, чітко описаний рецепт приготування блюда також є алгоритмом, у такому разі виконавцем є людина.

Визначення алгоритму

Єдиного «дійсного» визначення поняття «алгоритм» немає.

«Алгоритм — це кінцевий набір правив, який визначає послідовність операцій для вирішення конкретної безлічі завдань і володіє п'ятьма важливими рисами: кінцівка, визначеність, введення, вивід, ефективність». (Д. Э. Батіг)

«Алгоритм — це всяка система обчислень, що виконуються по строго певних правилах, яка після якого-небудь числа кроків свідомо приводить до рішення поставленої задачі». (А. Колмогоров)

«Алгоритм — це точне розпорядження, що визначає обчислювальний процес, що йде від варійованих початкових даних до шуканого результату». (А. Марков)

«Алгоритм — точне розпорядження про виконання в певному порядку деякої системи операцій, ведучих до вирішення всіх завдань даного типу». (Філософський словник / Під ред. М. М. Розенталя)

«Алгоритм — строго детермінована послідовність дій, що описує процес перетворення об'єкту з початкового стану в кінцевий, записана за допомогою зрозумілих виконавцеві команд». (Микола Дмитрович Угріновіч, підручник «Інформатика і информ. технології»)

«Алгоритм — це послідовність дій, направлених на отримання певного результату за кінцеве число кроків».

«Алгоритм — однозначно, доступно і стисло (умовні поняття — назви етапу) описана послідовність процедур для відтворення процесу з обумовленим завданням алгоритму результатом за заданих початкових умов. Універсальність (або спеціалізація) алгоритму визначається застосовністю і надійністю даного алгоритму для вирішення нестандартних завдань».

«Алгоритм — це зрозумілі і точні розпорядження виконавцеві зробити кінцеве число кроків, направлених на рішення поставленої задачі».

«Алгоритм — це деякий кінцевий набір розрахованих на певного виконавця операцій в результаті виконання яких через певне число кроків може бути досягнута поставлена мета або вирішено завдання певного типу».

«Алгоритм — це послідовність дій, що або приводить до рішення задачі, або пояснює чому це рішення отримати не можна».

«Алгоритм — це точна, однозначна, кінцева послідовність дій, яку повинен виконати користувач для досягнення конкретної мети або для вирішення конкретного завдання або групи завдань».

«Алгоритм — це точне розпорядження, яке задає обчислювальний (алгоритмічний) процес, що починається з довільного початкового даного і напрвленный на отримання повністю визначуваним цим початковим даним результату».

Формальні ознаки алгоритмів

Різні визначення алгоритму в явній або неявній формі містять наступний ряд загальних вимог:

Дискретність — алгоритм повинен представляти процес рішення задачі як послідовне виконання деяких простих кроків. При цьому для виконання кожного кроку алгоритму потрібний кінцевий відрізок часу, тобто перетворення початкових даних в результат здійснюється в часі дискретно.

Детермінована — визначеність. У кожен момент часу наступний крок роботи однозначно визначається станом системи. Таким чином, алгоритм видає один і той же результат (відповідь) для одних і тих же початкових даних. У сучасному трактуванні у різних реалізацій одного і того ж алгоритму повинен бути ізоморфний граф. З іншого боку, існують імовірнісні алгоритми, в яких наступний крок роботи залежить від поточного стану системи і випадкового числа, що генерується. Проте при включенні методу генерації випадкових чисел в список «початкових даних», імовірнісний алгоритм стає підвидом звичайного.

Зрозумілість — алгоритм для виконавця повинен включати тільки ті команди, які йому (виконавцеві) доступні, які входять в його систему команд.

Завершаємость (кінцівка) — при коректно заданих початкових даних алгоритм повинен завершувати роботу і видавати результат за кінцеве число кроків. З іншого боку, імовірнісний алгоритм може і ніколи не видати результат, але вірогідність цього рівна 0.

Масовість — алгоритм повинен бути застосовний до різних наборів початкових даних.

Результативність — завершення алгоритму певними результатами.

Алгоритм містить помилки, якщо приводить до отримання неправильних результатів або не дає результатів зовсім.

Алгоритм не містить помилок, якщо він дає правильні результати для будь-яких допустимих початкових даних

Історія терміну

Сучасне формальне визначення алгоритму було дане в 30—50-х роки XX століття в роботах Тюрінга, Поста, Черча (теза Черча — Тюрінга), Н. Вінера, А. А. Маркова.

Само слово «алгоритм» відбувається від імені ученого Абу Абдуллах Мухаммеда ибн Муса аль-Хорезми. Близько 825 року він написав твір, в якому вперше дав опис придуманої в Індії позиційної десяткової системи числення. На жаль, арабський оригінал книги не зберігся. Аль-хорезмі сформулював правила обчислень в новій системі і, ймовірно, вперше використовував цифру 0 для позначення пропущеної позиції в записі числа (її індійську назву араби перевели як as-sifr або просто sifr, звідси такі слова, як «цифра» і «шифр»). Приблизно в цей же час індійські цифри почали застосовувати і інші арабські учені. У першій половині XII століття книга аль-Хорезми в латинському перекладі проникла до Європи. Перекладач, ім'я якого до нас не дійшло, дав їй назву Algoritmi de numero Indorum («Алгорітмі про рахунок індійському»). Арабський же книга іменувалася Китаб аль-джебр валь-мукабала («Книга про складання і віднімання»). З оригінальної назви книги походить слово Алгебра.

Таким чином, ми бачимо, що латинізоване ім'я середньоазіатського ученого було винесене в заголовок книги, і сьогодні ні у кого немає сумнівів, що слово «алгоритм» потрапило в європейські мови саме завдяки цьому твору. Проте питання про його сенс тривалий час викликало запеклі спори. Впродовж багатьох століть походженню слова давалися самі різні пояснення.

Одні виводили algorism з грецьких algiros (хворий) і arithmos (число). З такого пояснення не дуже ясно, чому числа саме «хворі». Або ж лінгвістам хворими здавалися люди, що мають нещастя займатися обчисленнями? Своє пояснення пропонував і енциклопедичний словник Брокгауза і Ефрона. У нім алгорифм (до речі, до революції використовувалося написання алгоридм, через фиту) проводиться «від арабського слова Аль-горетм, тобто корінь». Зрозуміло, ці пояснення навряд чи можна визнати переконливими.

Згаданий вище переклад твору аль-Хорезми став першою ластівкою, і протягом декількох наступних сторіч з'явилася безліч інших праць, присвячених все тому ж питанню — навчанню мистецтву рахунку за допомогою цифр. І всі вони в назві мали слово algoritmi або algorismi.

Про аль-Хорезми пізніші автори нічого не знали, але оскільки перший переклад книги починається словами: «Dixit algorizmi: .» («Аль-хорезмі говорив: .»), все ще пов'язували це слово з ім'ям конкретної людини. Дуже поширеною була версія про грецьке походження книги. У англо-норманнской рукописі XIII століття, написаному у віршах, читаний:

"Алгорізм був придуманий в Греції. Це частина арифметики. Придуманий він був майстром на ім'я Алгорізм, який дав йому своє ім'я. І оскільки його звали Алгорізм, Він назвав свою книгу «Алгорізм».

Близько 1250 року англійський астроном і математик Іоанн Сакробосько написав працю по арифметиці «Algorismus vulgaris», на сторіччя що став основним підручником по обчисленнях в десятковій позиційній системі числення в багатьох європейських університетах. У введенні Сакробосько назвав автором науки про рахунок мудреця на ім'я Алгус (Algus). А в популярній середньовічній поемі «Роман про троянду» (1275—1280) Жана де Міна «грецький філософ Алгус» ставиться в один ряд з Платоном, Арістотелем, Евклідом і Птолемеєм! Зустрічався також варіант написання імені Аргус (Argus). І хоча, згідно старогрецької міфології, корабель «Арго» був побудований Ясоном, саме цьому Арго приписувалося будівництво корабля.

«Майстер Алгус» (або Аргус) став в середньовічній літературі уособленням рахункового мистецтва. І у вже згадуваній «Поемі про троянду», і у відомій італійській поемі «Квітка», написаною Дуранте, є фрагменти, в яких мовиться, що навіть «Mestre Argus» не зуміє підрахувати, скільки разів сваряться і миряться закохані. Великий англійський поет Джефрі Чосер в поемі «Книга герцогині» (1369 р.) пише, що навіть «славний лічильник Аргус» (noble countour Argus) не зможе визнати чудовисьок, що з'явилися в кошмарних баченнях героєві.

Втім, грецька версія була не єдиною. Міфічний Алгор (Algor) іменувався то королем Кастілії (Rex quodam Castelliae), то індійським королем, то арабським мудрецем (philosophus Algus nomine Arabicus).

Проте з часом такі пояснення все менш займали математиків, і слово algorism (або algorismus), незмінно присутнє в назвах математичних творів, знайшло значення способу виконання арифметичних дій за допомогою арабських цифр, тобто на папері, без використання абака. Саме у такому значенні воно увійшло в багато європейських мов. Наприклад, з позначкою «устар.» воно присутнє в представницькому словнику англійської мови Webster's New World Dictionary, виданому в 1957 р.

Алгоритм — це мистецтво рахунку за допомогою цифр, але спочатку слово «цифра» відносилося тільки до нуля. Знаменитий французький трувер Готье де Куанси (Gautier de Coincy, 1177—1236) в одному з віршів використовував слова algorismus-cipher (які означали цифру 0) як метафору для характеристики абсолютно нікчемної людини. Очевидно, розуміння такого образу вимагало відповідної підготовки слухачів, а це означає, що нова система числення вже була їм досить добре відома.

Багато століть абак був фактично єдиним засобом для практичних обчислень, їм користувалися і купці, і міняйла, і учені. Достоїнства обчислень на рахунковій дошці роз'яснював в своїх творах такий видатний мислитель, як Герберт Аврілакський (938—1003), що став в 999 р. татом римським під ім'ям Сильвестра II. Нове з величезною працею пробивало собі дорогу, і до історії математики увійшло наполегливе протистояння таборів абацистов і алгорисмиков (перших іноді ще називали гербекистами), які пропагували використання для обчислень абака замість арабських цифр. Цікаво, що відомий французький математик Николя Шюке (Nicolas Chuquet, 1445—1488) в реєстр платників податків міста Ліона був вписаний як алгорисмик (algoriste). Але пройшло не одне сторіччя, перш ніж новий спосіб рахунку остаточно утвердився, стільки часу було потрібно, щоб виробити загальновизнані позначення, удосконалити і пристосувати до запису на папері методи обчислень. У Західній Європі вчителів арифметики аж до XVII століття продовжували називати «магістрами абака», як, наприклад, математика Никколо Тарталью (1500—1557).

Отже, твору по мистецтву рахунку називалися Алгоритмами. З багатьох сотень можна виділити і такі незвичайні, як написаний у віршах трактат «Carmen de Algorismo» (латинське carmen і означає вірші) Олександра де Вілла Дєї (Alexander de Villa Dei, розум. 1240) або підручник віденського астронома і математика Георга Пурбаха (Georg Peurbach, 1423—1461) «Opus algorismi jocundissimi» («Веселий твір по алгоритму»).

Поступове значення слова розширювалося. Учені починали застосовувати його не тільки до суто обчислювальним, але і до інших математичних процедур. Наприклад, близько 1360 р. французький філософ Микола Кричимо (Nicolaus Oresme, 1323/25-1382) написав математичний трактат «Algorismus proportionum» («Обчислення пропорцій»), в якому вперше використовував ступені з дробовими показниками і фактично впритул підійшов до ідеї логарифмів. Коли ж на зміну абаку прийшов так званий рахунок на лініях, численне керівництво по ньому почало називати «Algorithmus linealis», тобто правила рахунку на лініях.

Можна звернути увагу на те, що первинна форма algorismi через якийсь час втратила останню букву, і слово придбало зручніше для європейської вимови вид algorism. Пізніше і воно, у свою чергу, піддалося спотворенню, швидше за все, пов'язаному із словом arithmetic.

У 1684 році Готфрід Лейбніц в творі «Nova Methodvs pro maximis et minimis, itemque tangentibus.» вперше використовував слово «алгоритм» (Algorithmo) в ще ширшому сенсі: як систематичний спосіб вирішення проблем диференціального числення.

У XVIII столітті в одному з німецьких математичних словників, Vollstandiges mathematisches Lexicon (виданому в Лейпцігу в 1747 р.), термін algorithmus все ще пояснюється як поняття про чотири арифметичні операції. Але таке значення не було єдиним, адже термінологія математичної науки в ті часи ще тільки формувалася. Зокрема, вираз algorithmus infinitesimalis застосовувався до способів виконання дій з нескінченно малими величинами. Користувався словом алгоритм і Леонард Ейлер, одна з робіт якого так і називається — «Використання нового алгоритму для вирішення проблеми Пелля» («De usu novi algorithmi in problemate Pelliano solvendo»). Ми бачимо, що розуміння Ейлером алгоритму як синоніма способу рішення задачі вже дуже близько до сучасного.

Проте було потрібно ще майже два сторіччя, щоб всі старовинні значення слова вийшли з вживання. Цей процес можна прослідкувати на прикладі проникнення слова «алгоритм» в російську мову.

Історики датують 1691 роком один із списків староруського підручника арифметики, відомого як «Рахункова мудрість». Цей твір відомий в багатьох варіантах (найраніші з них майже на сто років старше) і сходить до ще стародавніших рукописів XVI в. По ним можна прослідкувати, як знання арабських цифр і правил дій з ними поступово розповсюджувалося на Русі. Повна назва цього підручника — «Ця книга, глаголемая по еллински і по гречески арифметику, а по немецки алгоризма, а по русски цифирная рахункова мудрість».

Таким чином, слово «алгоритм» розумілося першими російськими математиками так само, як і в Західній Європі. Проте його не було ні в знаменитому словнику В. І. Даля, ні опісля сто років в «Тлумачному словнику російської мови» під редакцією Д. Н. Ушакова (1935 р.). Зате слово «алгорифм» можна знайти і в популярному дореволюційному Енциклопедичному словнику братів Гранат, і в першому виданні Великої Радянської Енциклопедії (БСЕ), виданому в 1926 р. І там, і там воно трактується однаково: як правило, по якому виконується те або інше з чотирьох арифметичних дій в десятковій системі числення. Проте до початку XX в. для математиків слово «алгоритм» вже означало будь-який арифметичний або алгебру процес, що виконується по строго певних правилах, і це пояснення також дається в БСЕ.

Алгоритми ставали предметом все більш пильної уваги учених, і поступове це поняття зайняло одне з центральних місць в сучасній математиці. Що ж до людей, від математики далеких, то на початок сорокових років це слово вони могли почути хіба що під час навчання в школі, в поєднанні «алгоритм Евкліда». Не дивлячись на це, алгоритм все ще сприймався як термін суто спеціальний, що підтверджується відсутністю відповідних статей в менш об'ємних виданнях. Зокрема, його немає навіть в десятитомній Малій Радянській Енциклопедії (1957 р.), не говорячи вже про однотомні енциклопедичні словники. Та зате опісля десять років, в третьому виданні Великої радянської енциклопедії (1969 р.) алгоритм вже характеризується як одна з основних категорій математики, що «не володіють формальним визначенням в термінах простіших понять, і що абстрагуються безпосередньо з досвіду». Як ми бачимо, відмінність навіть від трактування першим виданням БСЕ разюча! За сорок років алгоритм перетворився на одне з ключових

понять математики, і визнанням цього стало включення слова вже не в енциклопедії, а в словники. Наприклад, воно присутнє в академічному «Словнику російської мови» (1981 р.) саме як термін з області математики.

Одночасно з розвитком поняття алгоритму поступово відбувалася і його експансія з чистої математики в інші сфери. І почало їй поклало появу комп'ютерів, завдяки якій слово «алгоритм» увійшло в 1985 р. у всі шкільні підручники інформатики і знайшло нове життя. Взагалі можна сказати, що його сьогоднішня популярність безпосередньо пов'язана із ступенем розповсюдження комп'ютерів. Наприклад, в третьому томі «Дитячої енциклопедії» (1959 р.) про обчислювальні машини мовиться немало, але вони ще не стали чимось звичним і сприймаються швидше як якийсь атрибут світлого, але достатньо далекого майбутнього. Відповідно і алгоритми жодного разу не згадуються на її сторінках. Але вже на початку 70-х рр. минулого сторіччя, коли комп'ютери перестали бути екзотичною чудасією, слово «алгоритм» стрімко входить в ужиток. Це чуйно фіксують енциклопедичні видання. У «Енциклопедії кібернетики» (1974 р.) в статті «Алгоритм» він вже зв'язується з реалізацією на обчислювальних машинах, а в «Радянській військовій енциклопедії» (1976 р.) навіть з'являється окрема стаття «Алгоритм рішення задачі на ЕОМ». За останні півтора-два десятиліття комп'ютер став невід'ємним атрибутом наший життя, комп'ютерна лексика стає все більш звичною. Слово «алгоритм» в наші дні відоме, ймовірно, кожному. Воно упевнено зробило крок навіть в розмовну мову, і сьогодні ми нерідко зустрічаємо в газетах і чуємо у виступах політиків виразу ніби «алгоритм поведінки», «алгоритм успіху» або навіть «алгоритм зради». Академік Н. Н. Моїсєєв назвав свою книгу «Алгоритми розвитку», а відомий лікар Н. М. Амосов — «Алгоритм здоров'я» і «Алгоритми розуму». А це означає, що слово живе, збагачуючись все новими значеннями і смисловими відтінками.

Види алгоритмів

Особливу роль виконують прикладні алгоритми, призначені для вирішення певних прикладних завдань. Алгоритм вважається за правильне, якщо він відповідає вимогам завдання (наприклад, дає фізично правдоподібний результат). Алгоритм (програма) містить помилки, якщо для деяких початкових даних він дає неправильні результати, збої, відмови або не дає ніяких результатів взагалі. Остання теза використовується в олімпіадах по алгоритмічному програмуванню, щоб оцінити складені учасниками програми.

Важливу роль грають рекурсивні алгоритми (алгоритми, що викликають самі себе до тих пір, поки не буде досягнута деяка умова повернення). Починаючи з кінця XX - почала XXI століття активно розробляються паралельні алгоритми, призначені для обчислювальних машин, здатних виконувати декілька операцій одночасно.

Наявність початкових даних і деякого результату

Алгоритм — це точно певна інструкція, послідовно застосовуючи яку до початкових даних, можна отримати рішення задачі. Для кожного алгоритму є деяка безліч об'єктів, допустимих як початкові дані. Наприклад, в алгоритмі ділення дійсних чисел ділиме може бути будь-яким, а дільник не може бути рівний нулю.

Алгоритм служить, як правило, для вирішення не одного конкретного завдання, а деякого класу завдань. Так, алгоритм складання застосовний до будь-якої пари натуральних чисел. У цьому виражається його властивість масовості, тобто можливості застосовувати багато разів один і той же алгоритм для будь-якого завдання одного класу.

Для розробки алгоритмів і програм використовується алгоритмізація — процес систематичного складання алгоритмів для вирішення поставлених прикладних завдань. Алгоритмізація вважається обов'язковим етапом в процесі розробки програм і вирішенні завдань на ЕОМ. Саме для прикладних алгоритмів і програм принципово важлива детермінована, результативність і масовість, а також правильність результатів вирішення поставлених завдань.

Форма алгоритмів

Алгоритм може бути записаний словами і зображений схематично. Зазвичай спочатку (на рівні ідеї) алгоритм описується словами, але у міру наближення до реалізації він знаходить все більш формальні контури і формулювання на мові, зрозумілій виконавцеві (наприклад, машинний код). Наприклад, для опису алгоритму застосовуються блок-схеми. Іншим варіантом опису, не залежним від мови програмування, є псевдокод.

Ефективність алгоритмів

Хоча у визначенні алгоритму потрібна лише кінцівка числа кроків, потрібних для досягнення результату, на практиці виконання навіть хоч би мільярда кроків є дуже повільним. Також зазвичай є інші обмеження (на розмір програми, на допустимі дії). У зв'язку з цим вводять такі поняття як складність алгоритму (тимчасова я, за розміром програми, обчислювальна і ін.).

Для кожного завдання може існувати безліч алгоритмів, що приводять до мети. Збільшення ефективності алгоритмів складає одне із завдань сучасної інформатики. У 50-х рр. XX століття з'явилася навіть окрема її область — швидкі алгоритми. Зокрема, у відомій всім з дитинства завданню про множення десяткових чисел виявилися ряд алгоритмів, що дозволяють істотно (у асимптотичному сенсі) прискорити знаходження твору.

Більшість використовуваних у програмуванні алгоритмічних мов мають загальні риси. У той же час, при викладі ідеї алгоритму, наприклад, при публікації в науковій статті, не завжди доцільно користуватися якою-небудь конкретною мовою програмування, щоб не захаращувати виклад несуттєвими деталями. У таких випадках застосовується неформальна алгоритмічна мова, максимально наближений до природного. Мова такого типу називають псевдокодом. Для фахівця не становить праці переписати програму із псевдокоду на будь-яку конкретну мову програмування. Запис алгоритму на псевдокоді найчастіше ясніше й наочніше, вона дає можливість вільно вибирати рівень деталізації, починаючи від опису загалом і кінчаючи докладним викладом.

Алгоритмічні мови

Програмування починалося із запису програм безпосередньо у вигляді машинних команд (у кодах, як говорять програмісти). Пізніше для полегшення кодування була розроблена мова Асемблера, що дозволяє записувати машинні команди в символічному виді. Наприклад, програмістові не потрібно пам'ятати числовий код операції додавання, замість цього можна використати символічне позначення ADD. Мова Асемблера залежить від системи команд конкретного комп'ютера. Він досить зручний для програмування невеликих завдань, що вимагають максимальної швидкості виконання. Однак великі проекти розробляти мовою Асемблера важко. Головна проблема полягає в тому, що програма, написана на Асемблері, прив'язана до архітектури конкретного комп'ютера й не може бути перенесена на інші машини. При вдосконаленні комп'ютера всі програми на Асемблері доводиться переписувати заново.

Майже відразу з виникненням комп'ютерів були розроблені мови високого рівня, тобто мови, що не залежать від конкретної архітектури. Для виконання програми мовою високого рівня її потрібно спочатку перевести на мову машинних команд. Спеціальна програма, що виконує такий переклад, називається **транслятором** або **компілятором.** Після трансляція програма виконується безпосередньо комп'ютером. Існує також можливість перекладу програми на проміжну мову, що не залежить від архітектури конкретного комп'ютера, але проте максимально наближений до мови машинних команд. Потім програма проміжною мовою виконується спеціальною програмою, що називається **інтерпретатором**. Можливий також варіант **компіляції** (Just In Time Compilation), коли виконуваний фрагмент програми переводиться із проміжної мови на мову машинних команд безпосередньо перед виконанням.

Найбільше розповсюджені компілюючі мови - це Сі, С++, Фортран, Паскаль. Інтерпретуємі і Just In Time Compilation - це в основному об’єктно-орієнтовані мови, такі як Java, Visual Basic й C#. Всі вони спочатку переводяться на проміжну мову: для Java це так званий байткод мови Java, для Visual Basic й C# - так звана проміжна мова (Intermediate Language або просто IL), що є одним з основних компонентів платформи ".Net" фірми Microsoft. Проміжна мова може інтерпретуватися спеціальним виконавцем (наприклад, віртуальною Java-машиною), але, як правило, у сучасних системах застосовується компіляція Just In Time Compilation, що дозволяє досягти більшої швидкодії.

Історично одним з перших мов високого рівня був Фортран. Він виявився винятково вдалим - простим й у той же час дуже ефективним. Дотепер більша частина наукових й інженерних програм написана на Фортрані. Проте, в останні 20 років програмісти віддають перевагу мові Сі й пов'язаної з ним лінії об’єктно-орієнтованих мов - С++, Java й C#.

Іншою значною віхою в історії алгоритмічних мов є розробка мови Алгол-60 (розшифровується як алгоритмический мова - ALGOrithmic Language). Виникнення мови Алгол-60 пов'язане з розвитком структурного підходу до програмування, у якому використається вкладення конструкцій мови друг у друга. Так, основна одиниця мови - оператор - може бути простим або складовим, тобто складається у свою чергу з декількох операторів, укладених у блок за допомогою ключових слів begin й end. Усередині блоку можна описувати локальні змінні, недоступні ззовні блоку, і навіть підпрограми або функції.

Мова Алгол-60 сприяв розвитку алгоритмічних мов, його спадкоємцем є, наприклад, Паскаль і вся лінія пов'язаних з ним мов: Modula-2, Oberon й Delphi. Проте, Алгол-60 виявився далеко не таким вдалим, як Фортран. У ньому були присутні непродумані рішення, зокрема, можливість вкладення підпрограм усередину інших підпрограм, а також невдалий механізм передачі параметрів підпрограм. Через цього Алгол-60 не був реалізований на практиці в повній відповідності зі стандартом. Мова Паскаль з'явилася теж як корекція Алгола-60, але, на жаль, успадкував його головне невдале рішення - вкладеність підпрограм друг у друга. Також у первісному варіанті мови Паскаль була відсутня можливість розбивки програми на файли. Ці недоліки були потім виправлені автором Паскаля, чудовим швейцарським ученим і педагогом Ніклаусом Віртом, у мовах Modula-2 й Oberon. Але, на жаль, програмістське співтовариство проігнорувало мову Oberon, зупинившись на небагато поліпшеному варіанті мови Паскаль. У цей час Паскаль, як правило, використається для навчання програмуванню, але не в практичній роботі.

Нарешті, сама успішна мова програмування - мова Сі й пов'язана з ним лінія об’єктно-орієнтованих мов: С++, Java, C#. На відміну від Алгола-60, мова Сі був створений не теоретиками, а практичними програмістами, що володіють при цьому високою математичною культурою. Мова була розроблена наприкінці 60-х років XX століття. Він уперше дозволив реально позбутися від Асемблера при створенні

операційних систем. Наприклад, практично весь текст операційної системи Unix написаний мовою Сі й, таким чином, не залежить від конкретного комп'ютера. Головними перевагами Сі є його простота й відсутність псевдонаукових рішень (таких, як вкладеність блоків програм друг у друга: у Сі функція не може містити усередині себе іншу функцію, а змінні чітко розділяються на глобальні й локальні - не так, як в Алголі, де локальні змінні підпрограми є глобальними для всіх вкладених у неї підпрограм). Просто і ясно описаний механізм передачі параметрів у функцію (тільки за значенням). Програміст, що створює програму на Сі, завжди чітко розуміє, як ця програма буде виконуватися. Поняття покажчика, статичні й автоматичні (стекові) змінної мови Сі максимально близько відбивають пристрій будь-якого сучасного комп'ютера, тому програми на Сі ефективні й зручні для налагодження.

У наш час більшість програм пишеться на мовах Сі , C++, C#. Інтерфейс будь-якої операційної системи (так званий API - Application Program Interface), тобто набір системних викликів, призначених для розроблювачів прикладних програм, як правило, являє собою набір функцій мовою Сі. Нарешті, сучасні об’єктно-орієнтовані мови також засновані мовою Сі. Це мова C++, що займає проміжне положення між традиційними й об’єктно-орієнтованими мовами, а також об’єктно-орієнтовані мови Java й C#.

У курсі будемо використати псевдокод для неформального запису алгоритмів, а також мови C# для практичного програмування.

Послідовність рішення задачі по розробці програми

Послідовність рішення задачі по розробці програм складається з наступних етапів:

1. Формулювання задачі в термінах деякої прикладної області знань;
2. Формалізація задачі, побудова математичної та інформаційної моделі;
3. Розробка алгоритму рішення задачі;
4. Вибір мови програмування ;
5. Складання програми;
6. Тестування та відладка програми;
7. Передача програми в експлуатацію.

Формалізація задачі, побудова математичної та інформаційної моделі

Розробка програми починається з формалізації задачі , побудови математичної та інформаційної моделі. Запис задачі з допомогою математичних рівнянь, формулювання цілей рішення на основі математичних понять являється математичною постановкою задачі або формалізацією. Розробка математичної моделі, яка включає в себе:

- визначення вихідних даних;
- визначення кінцевого результату;
- визначення послідовності дій по перетворюванню початкових даних в кінцевий результат.

Задачі включають в себе математичні та інформаційні проблеми.

До математичних проблем можна віднести проблему адекватного опису задачі існуючими методами . До інформаційних проблем віднесемо необхідність збору та зберігання вхідної інформації.

Розробка алгоритму рішення задачі

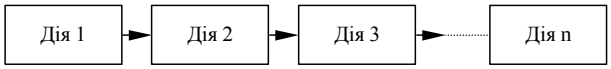
На основі розробленої математичної моделі створюється алгоритм, що являє собою послідовність дій, які описані в словесному, графічному, словесно-формульному вигляді. Алгоритм повинен відповідати шести основним властивостям: зрозумілість; дискретність; масовість; формальність; визначеність; закінченість.

Розробка програм забезпечується метода проектування зверху донизу - спочатку визначаємо задачу в загальних рисах, далі поступово уточнення цілей, алгоритмів, структур даних, обмежень і т п. шляхом врахування все більш мілких деталей задачі.

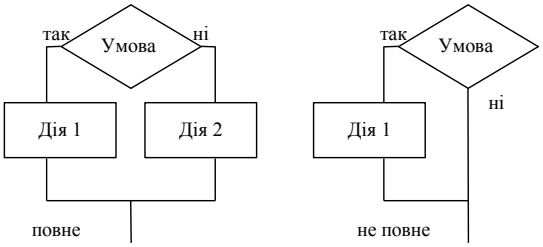
Метод поступового уточнення являється одним з основним методів розробки програм. Назвемо його «Нисходящим програмуванням »

Існує три базові алгоритмічні структури.

Перша - слідування.



Друга - розгалуження.



Третя- циклічні.

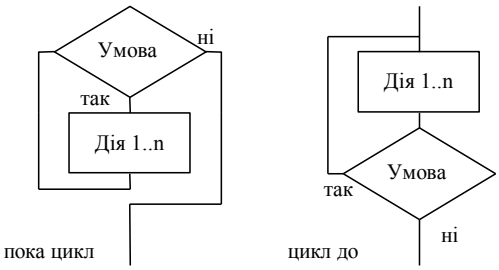


Рис 1. Базові алгоритмічні структури

Оскільки будь-яка програма є алгоритмічним рішенням поставленої задачі для ЕОМ, необхідно притримуватися деяких принципів, які, по-перше, сприяють написанню зрозумілих програм, по-друге, полегшують налагодження непрацюючих програм, по-третє, дозволяють зменшити загальну кількість логічних помилок. Будь яка багаторівнева структуризація - один з найважливіших дійових заходів в цьому напрямку. Виділяють такі рівні структуризації:

- синтаксичний;
- процедурний;
- модульний;
- об'єктно-компонентний.

Синтаксична структуризація визначає сучасну культуру написання програм і полягає в дотриманні декількох простих принципів:

- програма будується блоками, укладеними в основному між парою операторних дужок begin... end; такі блоки називаються складеними операторами;
- головним програмним блоком є тіло самої програми;
- запис програмних блоків проводиться за допомогою відповідних відступів від початку рядка; це поліпшує сприйняття структури програми;
- блоки можуть бути вкладеними;
- команди одного рівня укладеності записуються з однаковим відступом від початку рядка;
- команди одного рівня укладеності і близькі за призначенням іноді зручно записувати в одному рядку;
- розмір кожного програмного блоку бажано робити якомога меншим. Якщо якийсь програмний блок збільшується в розмірах більш ніж на 20 рядків або повторюється декілька разів в різноманітних місцях програми, то такий блок доцільно оформити в вигляді підпрограми (процедури або функції). Іноді дуже зручно розбити всю програму на окремі блоки, що оформляються в вигляді підпрограм. Наприклад: ввід даних, обчислення допоміжних значень, побудова графіків, вивід результатів на екран тощо.

Процедурна структуризація різко збільшує зрозумілість програми і дозволяє відпрацювати задачу окремими, легко доступними для огляду частинами.

Модульна структуризація використовується при побудові великих програм. в випадках, коли в програмі використовується велика кількість різноманітних процедур і функцій, усі підпрограми розбиваються на окремі групи, що оформляються, компілюються і зберігаються в вигляді окремих файлів-модулів. Часто з подібних модулів складаються цілі програмні бібліотеки. Щоб потім в основній програмі можна було використати процедури або функції з визначених модулів, необхідно після заголовка програми наступним рядком після ключового слова uses зазначити список усіх модулів, підпрограми яких ви збираєтеся використовувати.

Якщо компілятор виявляє в програмі незнайомий ідентифікатор, що не значиться в розділі описів поточної програми, то починається послідовний пошук цього імені в модулях, перерахованих в рядку uses ... Якщо він не знайдений і там, тоді видається помилка про те, що даний ідентифікатор невідомий.Оскільки модулі зберігаються в відкомпільованому вигляді, то при компіляції основної програми відбувається підстановка ділянки, що містить потрібні програми в модулях до коду основної програми.

Об'єктно-компонентний рівень припускає застосування особливої спеціалізованої ідеології і методології написання програм. Він передбачає достатньо високий рівень підготовки фахівців, що використовують дану технологію. При цьому уся програма складається з множини різноманітних об'єктів.

Навіть сама програма - це теж об'єкт. Будь-який об'єкт складається з набору властивостей і методів. Властивості визначають кількісні і якісні характеристики об'єктів, а методи - дії, які можуть виконувати ці об'єкти, тобто методи описують поводження об'єктів. Всі об'єкти функціонують в єдиному програмному середовищі, що знаходиться в постійному очікуванні якоїсь події. Такими подіями можуть бути натискання клавіші, клацання кнопки миші в визначеній області екрана, покращення курсору миші в конкретну область екрана тощо. Далі спеціальний програмний диспетчер переадресує цю подію для опрацювання потрібному об'єктові, в якого викликається для цих цілей відповідний метод. Об'єктно-орієнтована методологія дозволяє писати величезні програмні комплекси. Практично все сучасне програмне забезпечення (ОС Windoms, Word, Excel, Access тощо) написане з використанням даної технології.

Складання програми

Алгоритм описується операторами вибраної мови програмування.

Мова програмування - це набір ключових слів та певні правила їх запису, що забезпечують виконання послідовності дій відповідно до розробленого алгоритму.

Що ж входить в поняття мови програмування? По-перше, це набір дій, які ми можемо виконати з допомогою ЕОМ. Серед набору дій можна відмітити базові:

- Дія введення з різних пристроїв вводу (клавіатури, накопичувач на магнітних дисках та інші).
- Дія виведення на різні пристрої виводу (дисплей, накопичувач на магнітних дисках, принтери та інші).
- Дія присвоєння, коли ведеться обробка інформації в програмі.
- Дія переходу, коли змінюється послідовність виконання дій.

Названі дії дають можливість реалізувати першу базову алгоритмічну структуру. Для забезпечення реалізації другої базової структури мови програмування повинні мати засоби виконання дії при наявності умов. А для реалізації третьої алгоритмічної структури необхідно мати інструмент для повторення дій.

Тепер ми можемо відповісти на питання, що входить в поняття мови програмування:

1.Середовище програмування - це набір програм, що забезпечує комп'ютерний етап підготовки програми.

2.Форма представлення даних в програмі (константи, змінні, масиви і т.д.).

3. Набір ключових слів та певні правила їх запису, що забезпечують виконання послідовності дій відповідно до розробленого алгоритму.

Тестування та відладка програми

Відладка програми – це діяльність, направлена на виявлення і виправлення помилок в програмі з використанням процесів виконання його програмних модулів. *Тестування програми* – це процес виконання програми на деякому наборі даних, для якого заздалегідь відомий результат застосування або відомі правила поведінки цієї програми. Вказаний набір даних називається *тестовим* або просто *тестом*. Таким чином, відладку можна представити у вигляді багатократного повторення трьох процесів: тестування, в результаті якого може бути констатоване наявність в програмі помилки, пошуку місця помилки в програмах і документації і редагування програм і документації з метою усунення виявленої помилки. Іншими словами:

Відладка = Тестування + Пошук помилок + Редагування.

Успіх відладки програми в значній мірі зумовлює раціональна організація тестування. При відладці програми відшукуються і усуваються, в основному, ті помилки, наявність яких в програмі встановлюється при тестуванні. Як було вже відмічено, тестування не може довести правильність програми, в кращому разі воно може продемонструвати наявність в нім помилки. Виникає два завдання. Перше завдання: підготувати такий набір тестів, щоб виявити в нім по можливості більше число помилок. Проте чим довше продовжується процес тестування (і відладки в цілому), тим більшою стає вартість програми. Звідси друге завдання: визначити момент закінчення відладки програми (або окремих компонентів). Ознакою можливості закінчення відладки є повнота обхвату пропущеними через програму тестами безлічі різних ситуацій, що виникають при виконанні програм, і відносний рідкісний прояв помилок на останньому відрізку процесу тестування.

Для оптимізації набору тестів, тобто для підготовки такого набору тестів, який дозволяв би при заданому їх числі (або при заданому інтервалі часі, відведеному на тестування) виявляти більше число помилок, необхідно, по-перше, заздалегідь планувати цей набір і, по-друге, використовувати раціональну стратегію планування (проектування) тестів. Проектування тестів можна починати відразу ж після завершення етапу зовнішнього опису програми. Можливі різні підходи до вироблення стратегії проектування тестів, які можна умовно графічно розмістити (див. рис. 2) між наступними двома крайніми підходами . Лівий крайній підхід полягає в тому, що тести проектуються тільки на підставі вивчення специфікацій (зовнішнього опису, опису архітектури і специфікації модулів). Будова модулів при цьому ніяк не враховується, тобто вони розглядаються як чорні ящики. Фактично такий підхід вимагає повного

перебору всіх наборів вхідних даних, оскільки інакше деякі ділянки програм можуть не працювати при пропуску будь-якого тесту, а це означає, що помилки, що містяться в них, не виявлятимуться. Проте тестування повним безліччю наборів вхідних даних практично нездійсненно. Правий крайній підхід полягає в тому, що тести проектуються на підставі вивчення текстів програм з метою протестувати всі шляхи виконання програми. Якщо взяти до уваги наявність в програмах циклів із змінним числом повторень, то різних шляхів виконання програм може опинитися також надзвичайно багато, так що їх тестування також буде практично нездійсненно.

Рис. 2. Спектр підходів до проектування тестів

Оптимальна стратегія проектування тестів розташована усередині інтервалу між цими крайніми підходами, але ближче до лівого краю. Вона включає проектування значної частини тестів по специфікаціях, але вона вимагає також проектування деяких тестів і по текстах програм. При цьому в першому випадку ця стратегія базується на принципах:

- на кожну використовувану функцію або можливість – хоч би один тест
- на кожну область і на кожну межу зміни якої-небудь вхідної величини – хоч би один тест
- на кожну особливу (виняткову) ситуацію, вказану в специфікаціях, – хоч би один тест.

У другому випадку ця стратегія базується на принципі: кожна команда кожної програми повинна пропрацювати хоч би на одному тесті.

У цьому розділі даються загальні рекомендації по організації відладки програм. Але спочатку слід зазначити деякий феномен, який підтверджує важливість попередження помилок на попередніх етапах розробки: у міру зростання числа виявлених і виправлених помилок в програмі *росте* також відносна вірогідність існування в нім невиявлених помилок. Це пояснюється тим, що при зростанні числа виявлених помилок уточнюється і наше уявлення про загальне число допущених в нім помилок, а значить, в якійсь мірі, і про число невиявлених ще помилок.

Нижче приводяться рекомендації по організації відладки у формі заповідей.

Заповідь 1. Рахуйте тестування ключовим завданням розробки програми, доручайте його найкваліфікованішим і обдарованішим програмістам; небажано тестувати свою власну програму.

Заповідь 2. Хороший той тест, для якого висока вірогідність виявити помилку, а не той, який демонструє правильну роботу програми.

Заповідь 3. Готуйте тести як для правильних, так і для неправильних даних.

Заповідь 4. Документуйте пропуск тестів через комп'ютер; детально вивчайте результати кожного тесту; уникайте тестів, пропуск яких не можна повторити.

Заповідь 5. Кожен модуль підключайте до програми тільки один раз; ніколи не змінюйте програму, щоб полегшити її тестування.

Заповідь 6. Пропускайте наново всі тести, пов'язані з перевіркою роботи якої-небудь програми або її взаємодії з іншими програмами, якщо до неї були внесені зміни (наприклад, в результаті усунення помилки).

Задача «Sprite»

Група програмістів зібралася в понеділок і на всі свої гроші купила «Sprite» в пляшках місткістю по 0,25 л., не забувши узяти здачу. У вівторок вони здали порожній посуд, додали здачу, що залишилася, і знов купили стільки таких же пляшок «Sprite», скільки могли. Так вони діяли до п'ятниці. В п'ятницю, здавши посуд і додавши здачу з четверга, вони змогли купити тільки одну пляшку напою. При цьому грошей у них вже не залишилося. Вам необхідно написати програму, яка визначає мінімальну суму, яку мали програмісти в понеділок.

Формат вхідних даних: Вхідний файл складається з єдиного рядка, що містить два цілі числа F (вартість однієї пляшки «Sprite») і P (вартість однієї порожньої пляшки з під «Sprite») розділених пропуском.

Обмеження на вхідні дані

1) Вартості пляшки «Sprite» і порожньої пляшки виражаються цілими позитивними числами і не перевершують 1 000 000 коп. кожна;
2) Порожня пляшка дешевша за повну пляшку;
3) Початкова сума не перевершує 2 000 000 000 коп.

Формат вихідних даних:
У вихідний файл виводиться число, що визначає обчислену кількість грошей в копійках
Приклади вхідних і вихідних даних:

Вхідні дані:	Вихідні дані:
7 3	83

Розв‘язок задачі на мові C#:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
namespace Sprite
{
   class Sprite
   {
      static void Main()
      {
         StreamReader IN = new StreamReader("a.in");
         string s = IN.ReadLine();
         char[] c = s.Split(' ');
         int a, b, k, s, x;
         a = int.Parse(s[0]);
         b = int.Parse(s[1]);
         x = a; k = 0;
         while ((k != 1)& (x<2000000))
         {
            k = x / a;
            s = x - k * a;
            for (int i = 1; i <= 4; i++)
            {
               s = s + k * b;
               k = s / a;
               s = s - k * a;
            }
            x++;
         }
         StreamWriter OUT = new StreamWriter("a.out");
         x = x-1;
         OUT.WriteLine(x.ToString);
         OUT.Close();
      }
   }
}
```

Тести:

a.in	a.out
7 3	83
2 1	17
9 5	41
8 4	68
10 4	136
250 25	250225
12 3	777
17 6	303
150 10	506390
98 12	52128

Тестуюча програма:

```
@echo off
for %%t in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) do (
ECHO ---TEST %%t---
copy DATS\a%%t.in a.in >nul
```

```
a.exe
fc /w a.out SOLS\a%%t.out
pause
)
del a.exe
del a.in
del a.out
pause
```

Якщо для Вас приведений текст програми невідомий, пропустіть і поверніться до нього після опрацювання третього модуля курсу.

Я бажаю Вам успіхів у вивченні нової науки для Вас : АЛГОРИТМІЗАЦІЇ І ПРОГРАМУВАННЯ.

Література

Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1

Дональд Кнут Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol.1. Fundamental Algorithms. — 3-е изд. — М.: «Вильямс», 2006. — С. 720. — ISBN 0-201-89683-4

Порублев Илья Николаевич, Ставровский Андрей Борисович Алгоритмы и программы. Решение олимпиадных задач. — М.: «Вильямс», 2007. — С. 480. — ISBN 978-5-8459-1244-2

http://ru.wikipedia.org/wiki/Алгоритм