ЛЕКЦІЯ 2

ЕЛЕМЕНТАРНІ СТРУКТУРИ ДАНИХ

2.1. Структурування і абстракція програм.

Будь-які достатньо великі програми проектуються шляхом декомпозиції задачі – виділення в задачі деяких структур і їхніх абстракцій. Абстрагування від проблеми передбачає ігнорування ряду деталей для того, щоб звести задачу до більш простої. Задачі абстрагування і наступна декомпозиція типові для процесу створення програм: декомпозиція використовується для поділу програм на компоненти; абстрагування ж передбачає продуманий вибір компонентів для цієї задачі.

№2

Структурний підхід до даних і алгоритмів дає можливість структурування складної програми. Розробляти сучасну програмну методом „все відразу” неможливо, вона повинна бути представлена в вигляді деякої структури – складових частин і зв’язків між ними. Правильне структурування дає можливість на кожному етапі розробки зосередити увагу розробника на одній оглядовій її частині або доручити реалізацію різних її частин різним виконавцям.

При структуруванні програм можна застосовувати підхід, який базується на структуризації алгоритмів і відомий, як „низхідне” проектування – „програмування з верху в низ”, або підхід, який базується на структуризації даних і відомий, як „висхідне” проектування – „програмування з низу в верх”.

У першому випадку – структуризації алгоритмів – структурують перш за все дії, які повинна виконувати програма. Велику і складну задачу представляють у вигляді декількох задач меншого обсягу. Таким чином, модуль самого верхнього рівня, який відповідає за вирішення задачі в цілому, отримується достатньо простим і він забезпечує тільки послідовність звертань до модулів, які вирішують задачі нижчого рівня. Потім кожна задача в свою чергу підлягає декомпозиції за такими ж правилами. Процес дроблення продовжується до тих пір, поки на черговому рівні декомпозиції не отримують задачу, реалізація якої буде досить простою. Для цього випадку будь-яку програму можна представити набором наступних функціональних абстракцій.

Аналізуючи таке представлення програми, можна отримати узагальнену абстракцію функції.

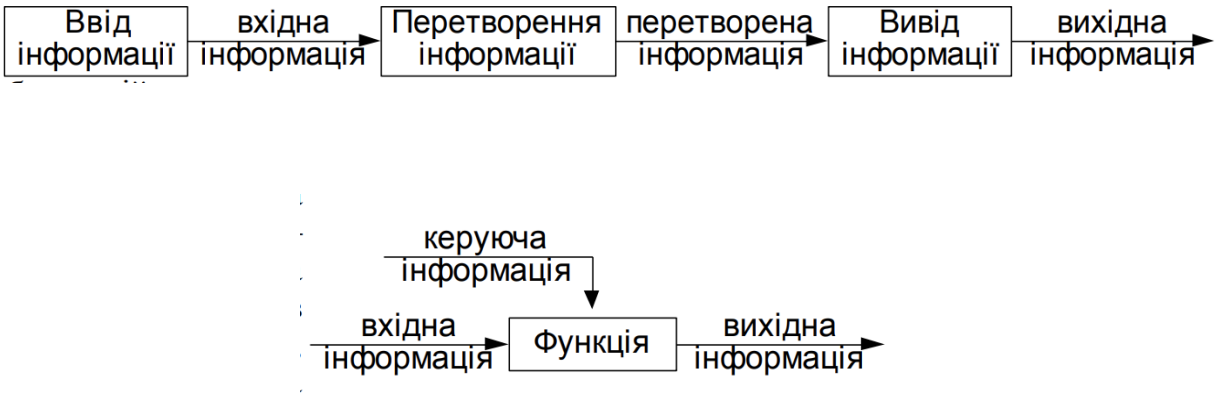


Рисунок 2.1 – Представлення структуризації алгоритмів

Інший підхід базується на структуризації даних (програмування – це обробка даних). У програмах можна застосовувати різні алгоритми, але в реальної програми завжди є замовник, який маючи вхідні дані, хоче, щоб за ними можна було отримати вихідні дані, а якими засобами це забезпечується – його не цікавить. Таким чином, задачею будь-якої програми є перетворення вхідних даних у вихідні. Мови програмування представляють набір базових типів даних і операції над ними. Інтегруючи базові типи, програміст створює більш складні типи даних і визначає нові операції над складними типами. В ідеалі останній крок композиції дає типи даних, які відповідають вхідним і вихідним даним задачі, а операції над цими типами реалізують в повному обсязі задачу проекту. Розглядаючи програму не як набір функцій, а перш за все, як деякий набір даних, кожен, із яких має дозволену групу функцій, отримують наступне абстрактне представлення програми

№3

Аналіз такого представлення програми дозволяє також отримати абстракцію даних.

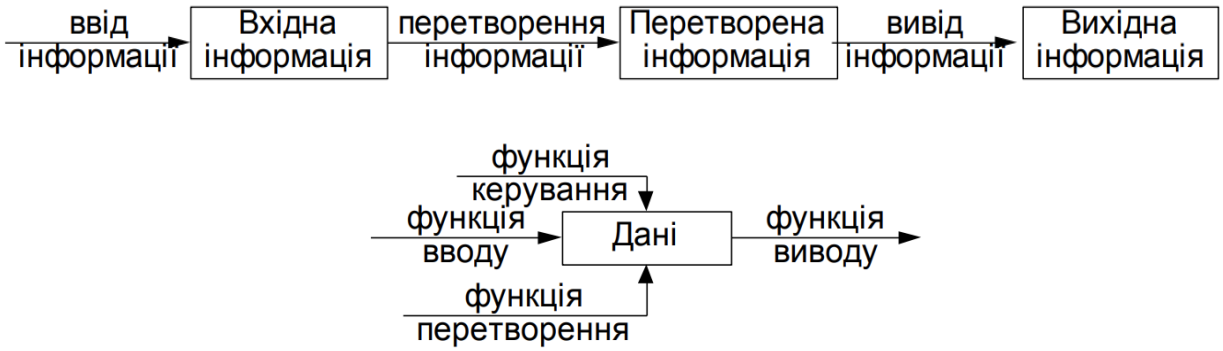


Рисунок 2.2 – Представлення структуризації даних

Отримані абстракції алгоритмів і даних лежать в основі багатьох формалізованих методів розробки програмного забезпечення.

Не можна протиставляти низхідне проектування висхідному і дотримуватись одного вибраного підходу. Реалізація будь-якого реального проекту завжди ведеться зустрічними шляхами, причому, з постійною корекцією структур алгоритмів за результатами розробки структур даних і навпаки.

№4

Ще одним досить продуктивним технологічним методом, який зв’язаний з структуризацією даних є інкапсуляція. Зміст її полягає у тому, що сконструйований новий тип даних – „будівельний блок” – оформляється таким чином, що його внутрішня структура стає недоступною для програміста – користувача цього типу. Програміст, який використовує цей тип даних у своїй програмі, може оперувати з даними цього типу тільки через виклик функцій, які визначені для цього типу. Новий тип даних представляється для нього у вигляді „чорного ящика” для якого відомі входи й виходи, але вміст – невідомий і недоступний.

Сучасні мови програмування блочного типу мають достатньо розвинуті можливості побудови програм з модульною структурою і управління доступом модулів до даних і функцій. Розширення ж мов додатковими можливостями конструювання типів і їх інкапсуляції робить мову об’єктно-орієнтованою. Сконструйовані й повністю закриті типи даних представляють собою класи, а функції, які працюють з їх внутрішньою структурою – методи

роботи з класами. При цьому в значній мірі міняється й сама концепція програмування. Програміст, який оперує об'єктами, вказує в програмі „що" потрібно зробити з об'єктом, а не „як" це потрібно зробити.

№5

1.2. Концепція структур даних

Структури даних і алгоритми є тими матеріалами, з яких будуються програми. Більше того, сам комп'ютер складається з структур даних і алгоритмів. Вбудовані структури даних представлені регістрами й словами пам'яті, де зберігаються бінарні величини. Закладені в конструкцію апаратури алгоритми – це реалізація в електронних логічних схемах жорстких правил, за якими поміщені в пам'ять дані інтерпретуються як команди, що підлягають виконанню. Тому в основі роботи будьякого комп'ютера лежить вміння оперувати лише з одним видом даних – з окремими бітами. Працює ж з цими даними комп'ютер тільки у відповідності з незмінним набором алгоритмів, які визначаються системою команд центрального процесора.

Задачі, які вирішуються за допомогою комп'ютера, рідко представляються мовою бітів. Як правило, дані мають форму чисел, літер, текстів, символів і більш складних структур типу послідовностей, списків і дерев.

Структура даних відноситься за своєю суттю до „просторових" понять: її можна звести до схеми організації інформації в пам'яті комп'ютера. Алгоритм ж є відповідним процедурним елементом в структурі програми – він служить рецептом розрахунку.

Структури даних, які застосовуються в алгоритмах, можуть бути досить складними. Вибір правильного представлення даних часто служить ключем до вдалого програмування і може в більшій мірі впливати на продуктивність програми, ніж деталі реалізації використовуваного алгоритму. Але, мабуть, ніколи не появиться загальна теорія вибору структур даних, у кожному конкретному випадку потрібно підходити до цього творчо.

Незалежно від змісту і складності будь-які дані в пам'яті комп'ютера представляються послідовністю бінарних розрядів, а їх значеннями є відповідні бінарні числа. Дані, які розглядаються у вигляді послідовності бітів,

мають дуже просту організацію, тобто є слабо структурованими. Більш крупні й змістовніші, ніж біт, „будівельні блоки” для організації довільних даних отримуються на основі поняття „структури даних”.

Поняття „фізична структура даних” відображає спосіб фізичного представлення даних в пам’яті машини і називається ще структурою зберігання.

Розгляд структури даних без врахування її представлення в машинній пам’яті називається абстрактною або логічною структурою. В загальному випадку між логічною і відповідною їй фізичною структурами існує відмінність, міра якої залежить від самої структури і особливостей того середовища, в якому вона повинна бути відображена.

№6

Розгляд структур даних утруднений досить незручним змішуванням абстрактних властивостей структур даних і проблемами представлення, які зв’язані з комп’ютером і машинною мовою. Важливо чітко розрізняти ці проблеми за допомогою багаторівневого опису. Будь-яка структура даних може описуватися, таким чином, на трьох різних рівнях:

- Функціональна специфікація – вказує для деякого класу імен операції, які дозволені з цими іменами, і властивості цих операцій;

- Логічний опис – задає декомпозицію об’єктів на більш елементарні об’єкти і декомпозицію відповідних операцій на більш елементарні операції;

- Фізичне представлення – дає метод розміщення в пам’яті комп’ютера тих величин, які складають структуру, і відношення між ними, а також спосіб кодування операцій на мові програмування.

Одній і тій же функціональній специфікації можуть відповідати декілька логічних описів, які в свою чергу можуть реалізовуватися декількома фізичними представленнями. Проте, потрібно мати впевненість, що декомпозиція кожного нового рівня достатньо добре відображає декомпозицію безпосередньо вищого рівня.

№7

Класифікація структур даних

Під структурою даних в загальному випадку розуміють множину елементів даних і множину зв’язків між ними.

Розрізняються прості (базові) структури даних і інтегровані (структуровані).

Простими називаються такі структури даних, які не можуть бути розділені на складові частини більші, ніж біти. З точки зору фізичної структури важливою є та обставина, що в даній машинній архітектурі, в даній системі програмування завжди можна наперед сказати, яким буде розмір даного простого типу і яка структура його розміщення в пам’яті. З логічної точки зору прості дані є неподільними одиницями.

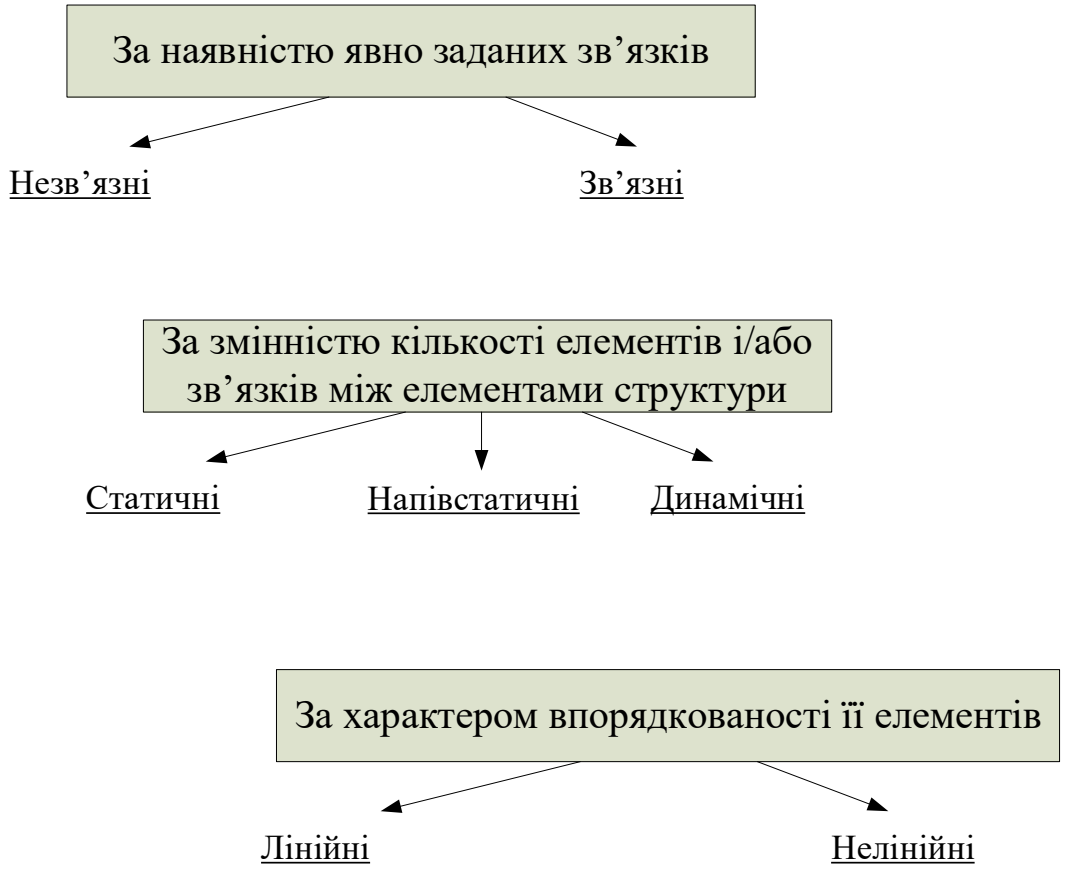
Інтегрованими називаються такі структури даних, складовими частинами яких є інші структури даних – прості але в свою чергу інтегровані. Інтегровані структури даних конструюються програмістом з використанням засобів інтеграції даних, які представляються мовами програмування.

№8

У залежності від наявності чи відсутності явно заданих зв’язків між елементами даних потрібно розрізняти незв’язні структури і зв’язні структури.

Досить важлива ознака структури даних – її змінність – зміна кількості елементів і/або зв’язків між елементами структури. За ознакою змінності розрізняють структури статичні, напівстатичні, динамічні.

Ще одна важлива ознака структури даних – характер впорядкованості її елементів. За цією ознакою структури можна поділити на лінійні й нелінійні структури.



№9

Операції над структурами даних

Над будь-якими структурами даних можуть виконуватися чотири загальні операції: створення, знищення, вибір (доступ), поновлення.

Операція створення полягає у виділенні пам’яті для зберігання структури даних. Пам’ять може виділятися в процесі виконання програми або на етапі компіляції. У деяких мовах для структурованих даних, які конструюються програмістом, операція створення включає в себе також встановлення початкових значень параметрів, створюваної структури (ініціалізація).

Незалежно від використовуваної мови програмування, наявні в програмі структури даних не появляються „з нічого”, а явно або неявно оголошуються операторами створення.

Операція знищення структур даних протилежна до операції створення – вона звільнює пам’ять, яку займала структура, для подальшого використання. Операція знищення дозволяє ефективно використовувати пам’ять.

Операція вибору використовується програмістами для доступу до даних в самій структурі. Форма операції доступу залежить від типу структури даних, до якої здійснюється звертання.

Метод доступу – одна з найбільш важливих властивостей структур, тому що вона має безпосереднє відношення до вибору конкретної структури даних.

Операція поновлення дозволяє змінити значення даних в структурі даних. Прикладом операції поновлення є операція присвоєння, або, більш складна форма – передача параметрів.

Вищевказані чотири операції обов'язкові для всіх структур і типів даних.

Крім цих загальних операцій для кожної структури даних можуть бути визначені специфічні операції, які працюють тільки з даними конкретного типу чи структури.

ПРОСТІ СТРУКТУРИ ДАНИХ

№10

Арифметичні типи

Стандартні типи даних часто називають арифметичними, оскільки їх можна використовувати в арифметичних операціях. Для опису основних типів визначено ключові слова: цілий, символьний, логічний, дійсний.

За допомогою цілих чисел можна представити кількість об'єктів, яка є дискретною за своєю природою (тобто кількість об'єктів можна перерахувати). Внутрішнє представлення величин цілого типу – ціле число в бінарному коді. Внутрішнє представлення дійсного типу складається з двох частин – мантиси і порядку.

Результати логічного типу отримуються при порівнянні даних будь-яких типів. Величини логічного типу можуть приймати тільки значення true і false. Внутрішня форма представлення значення false – 0 (нуль). Будь-яке інше значення інтерпретується як true.

Значенням символьного типу є символи з деякої наперед визначеної множини. В більшості сучасних персональних комп'ютерів цією множиною є ASCII. Ця множина складається з 256 різних символів, які впорядковані

певним чином і містить символи великих і малих букв, цифр і інших символів, включаючи спеціальні керуючі символи. Значення символьного типу займає в пам’яті 1 байт. Іншою широко використовуваною множиною для представлення символьних даних є код Unicode. В Unicode кожний символ кодується двома байтами.

Над арифметичними типами, як і над всіма іншими, можливі перш за все чотири основних операції: створення, знищення, вибір, поновлення. Специфічні операції над числовими типами – додавання, віднімання, множення і ділення.

Ще одна група операцій над арифметичними типами – операції порівняння: „рівно”, „не рівно”, „більше”, „менше” і т.п. Говорячи про операції порівняння, потрібно звернути увагу на особливість виконання порівнянь на рівність/нерівність дійсних чисел. Оскільки ці числа представляються в пам’яті з деякою точністю, порівняння їх не завжди може бути абсолютно достовірним.

№11

Перерахований тип

При написанні програм часто виникає потреба визначити декілька іменованих констант, для яких потрібно, щоб усі вони мали різні значення. Для цього зручно використовувати перерахований тип.

Перерахований тип представляє собою впорядкований тип даних, який визначається програмістом, тобто програміст перераховує всі значення, які може приймати змінна цього типу.

Діапазон значень перечислень визначається кількістю біт, які необхідні для представлення всіх його значень. Будь-яке значення цілого типу можна явно привести до перерахованого типу, але при виході за межі його діапазону результат буде невизначеним. При відсутності ініціалізації перша константа приймає нульове значення, а кожній наступній присвоюється на одиницю більше значення від попереднього.

На фізичному рівні над змінними перерахованого типу визначені операції створення, знищення, вибору, поновлення. При цьому виконується

визначення порядкового номера ідентифікатора за його значенням і, навпаки, за номером ідентифікатора його значення.

При виконанні арифметичних операцій перечислення перетворюються у ціле. Оскільки перечислення є типом, який визначається користувачем, для нього можна вводити власні операції.

№12

Покажчики

Тип „покажчик” представляє собою адресу комірки пам’яті (в більшості сучасних обчислювальних систем розмір комірки – мінімальної адресованої одиниці пам’яті – складає один байт). При програмуванні на низькому рівні робота з адресами складає значну частину програм. При вирішенні прикладних задач з використанням мов високого рівня найбільш часті випадки, коли програміст може використовувати покажчики, наступні:

1. При необхідності представити одну й ту ж ділянку пам’яті, а значить, одні й ті ж фізичні дані, як дані різної логічної структури.

2. При роботі з динамічними структурами даних.

В C++ розрізняють три види покажчиків – покажчик на об’єкт, на функцію і на пустий тип, які відрізняються властивостями і набором допустимих операцій. Покажчик не є самостійним типом, він завжди зв’язаний з якимось іншим типом.

У програмах на мовах високого рівня покажчики можуть бути типізованими і нетипізованими. При оголошені типізованого покажчика визначається й тип об’єкту в пам’яті, який адресується цим покажчиком.

Хоча фізична структура адреси не залежить від типу й значення даних, які зберігаються за цією адресою, компілятор вважає покажчики на різні типи такими, що мають різний тип. Таким чином, коли йде мова про типізовані покажчики, правильніше говорити не про єдиний тип даних „покажчик”, а про цілу сім’ю типів: „покажчик на ціле”, „покажчик на символ”.

Нетипізований покажчик використовується для представлення адреси, за якою містяться дані невідомого типу. Робота з нетипізованими

покажчиками суттєво обмежена, вони можуть використовуватися тільки для збереження адреси, звертатися за такою адресою не можна.

Основними операціями, в яких беруть участь покажчики є присвоєння, отримання адреси, вибір.

Перерахованих операцій достатньо для вирішення задач прикладного програмування тому набір операцій над покажчиками в більшості мов програмування цим й обмежується. Системне програмування потребує більш гнучкої роботи з адресами, тому в C/C++ доступні також операції адресної арифметики.

СТАТИЧНІ СТРУКТУРИ ДАНИХ

Статичні структури відносяться до класу структур, які представляють собою структуровану множину примітивних, базових, структур. Оскільки статичні структури відрізняються відсутністю змінності, пам'ять для них виділяється автоматично – як правило, на етапі компіляції, або при виконанні – в момент активізації того програмного блоку, в якому вони описані. Ряд мов програмування допускають розміщення статичних структур в пам'яті на етапі виконання за явною вимогою програміста, але й у цьому випадку обсяг виділеної пам'яті залишається незмінним до знищення структури. Виділення пам'яті на етапі компіляції є такою зручною властивістю статичних структур, що у ряді задач програмісти використовують їх навіть для представлення об'єктів, які мають властивість змінності. Наприклад, коли розмір масиву невідомий наперед, для нього резервується максимально можливий розмір.

Статичні структури в мовах програмування зв'язані із структурованими типами. Структуровані типи в мовах програмування є тими засобами інтеграції, які дозволяють будувати структури даних будь-якої складності. До таких типів відносяться масиви, структури та їхні похідні типи.

№13

Масиви

Логічно масив об'єднує елементи одного типу даних, тобто належить до однорідного типу даних. Більше формально його можна визначити як

впорядковану сукупність елементів деякого типу, які адресуються за допомогою одного або декількох індексів.

Масиви можна класифікувати за кількістю розмірностей масиву масиви поділяються на одновимірні масиви (вектори), двохвимірні (матриці) і багатовимірні (трьох, чотирьох і більше).

Логічно масив – це така структура даних, яка характеризується:

- фіксованим набором елементів одного і того ж типу;
- кожний елемент має унікальний набір значень індексів;
- кількість індексів визначають мірність масиву;
- звернення до елемента масиву виконується за ім'ям масиву і значенням індексів для даного елемента.

Фізична структура масиву – це спосіб розміщення елементів масиву в пам'яті комп'ютера. Під елемент масиву виділяється кількість байт пам'яті, яка визначається базовим типом елемента цього масиву. Кількість елементів масиву і розмір базового типу визначають розмір пам'яті для зберігання масиву.

Найважливіша операція фізичного рівня над масивом – доступ до заданого елемента. Як тільки реалізовано доступ до елемента, над ним може бути виконана будь-яка операція, що має сенс для того типу даних, якому відповідає елемент. Перетворення логічної структури масиву у фізичну називається процесом лінеаризації, в ході якого багатовимірна логічна структура масиву перетвориться в одновимірну фізичну структуру.

Адресою масиву є адреса першого байту початкового компоненту масиву. Індексація масивів в C/C++ обов'язково починається з нуля. До операцій логічного рівня над масивами необхідно віднести такі як сортування масиву, пошук елемента за ключем.

Розріджені масиви

На практиці зустрічаються масиви, які через певні причини можуть займати пам'ять не повністю, а частково. Це особливо актуально для масивів великих розмірів, таких що для їхнього зберігання в повному об'ємі пам'яті

може бути недостатньо. Розріджений масив – це масив, більшість елементів якого рівні між собою, так що зберігати в пам’яті достатньо лише невелику кількість значень відмінних від основного (фонового) значення інших елементів. При роботи з розрідженими масивами питання розташування їх в пам’яті реалізуються на логічному рівні з врахуванням їхнього типу.

Розрізняють два типи розріджених масивів:

- масиви, в яких розташування елементів із значеннями відмінними від фонового, можуть бути описані математично;
- масиви з випадковим розташуванням елементів.

До масивів з математичним описом розташування елементів відносяться масиви, в яких існує закономірності в розташуванні елементів із значеннями відмінними від фонового.

Елементи, значення яких є фоновими, називають нульовими; елементи, значення яких відмінні від фонового, – ненульовими. Фонове значення не завжди рівне нулю. Ненульові значення зберігаються, як правило, в одновимірному масиві, а зв’язок між розташуванням у розрідженому масиві і в новому, одновимірному, описується математично за допомогою формули, що перетворює індекси масиву в індекси вектора.

На практиці для роботи з розрідженим масивом розробляються функції:

- для перетворення індексів масиву в індекс вектора;
- для отримання значення елемента масиву з його упакованого представлення за індексами;
- для запису значення елемента масиву в її упаковане представлення за індексами.

До масивів з випадковим розташуванням елементів відносяться масиви, в яких не існує закономірностей у розташуванні елементів із значеннями відмінними від фонового.

Один з основних способів зберігання подібних розріджених матриць полягає в запам’ятовуванні ненульових елементів в одновимірному масиві і ідентифікації кожного елемента масиву індексами.

Дане представлення масиву скорочує вимоги до об’єму пам’яті більш ніж в 2 рази. Спосіб послідовного розподілу має також ту перевагу, що операції над матрицями можуть бути виконані швидше, ніж це можливо при представленні у вигляді послідовного масиву, особливо якщо розмір матриці великий.

Методи послідовного розміщення для представлення розріджених матриць звичайно дозволяють швидше виконувати операції над матрицями і більш ефективно використати пам’ять, ніж методи із зв’язаними структурами. Проте послідовне представлення матриць має певні недоліки. Так включення і виключення нових елементів матриці викликає необхідність переміщення великої кількості інших елементів. Якщо включення нових елементів і їхнє виключення здійснюється часто, то повинен бути вибраний метод зв’язаних структур.

Метод зв’язаних структур, проте, переводить структуру даних, що представляється, в інший розділ класифікації. При тому, що логічна структура даних залишається статичною, фізична структура стає динамічною.

№14

Множини

Тип даних „множина” не реалізований як стандартний тип мови програмування C/C++, але дуже часто використовується в програмуванні і реалізовується засобами визначення типів користувача, тому дамо йому короткий опис.

Множина – така структура, яка є набором даних одного і того ж типу, що не повторюються (кожен елемент множини є унікальним). Порядок слідування елементів множини на має принципового значення.

До множин застосовується стандартний принцип виключення. Це означає, що конкретний елемент або є членом множини, або ні. Множина може бути пустою, таку множину називають нульовою.

Множина є підмножиною іншої множини, якщо в цій другій множині можна знайти усі елементи, які є в першій множині. Відповідно, множина

вважається надмножиною іншої множини, якщо вона містить усі елементи цієї другої множини.

Кожен окремий елемент є членом множини, якщо він входить до складу елементів множини.

№15

Над множинами визначені наступні специфічні операції:

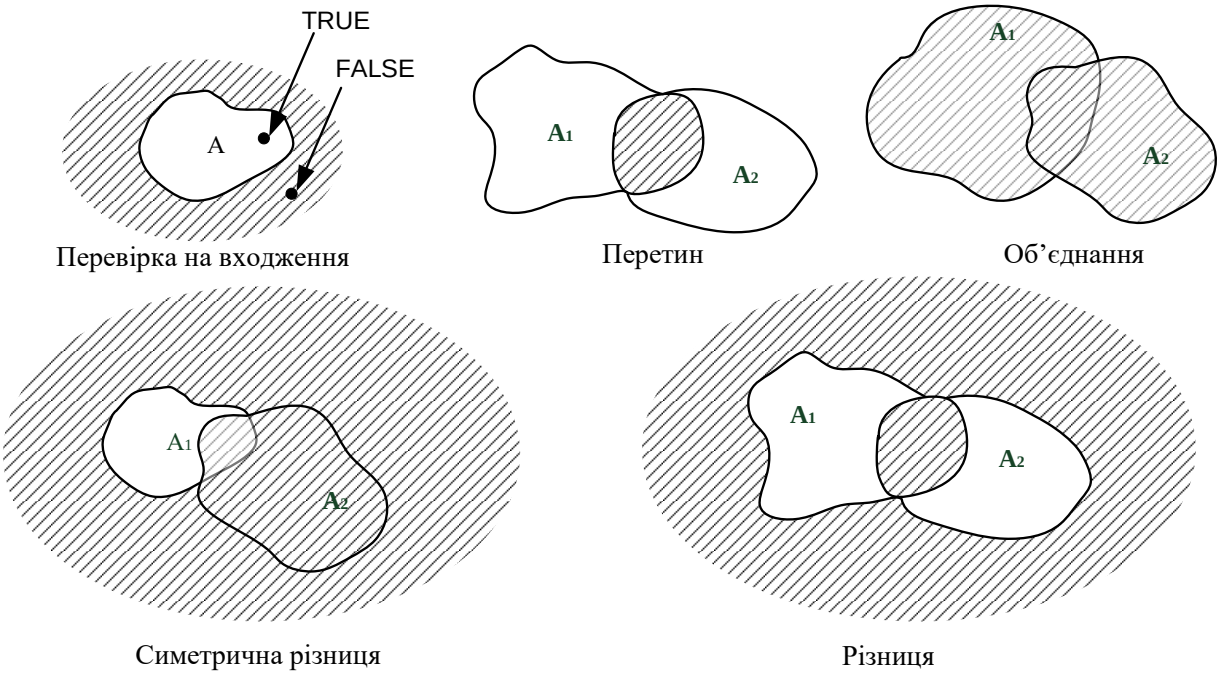
1. Об’єднання множин. Результатом є множина, що містить елементи початкових множин.

2. Перетин множин. Результатом є множина, що містить спільні елементи початкових множин.

3. Різниця множин. Результатом є множина, яка містить елементи першої множини, які не входять в другу множину.

4. Симетрична різниця. Результатом є множина, яка містить елементи, які входять до складу однієї або другої множини (але не обох).

5. Перевірка на входження елемента в множину. Результатом цієї операції є значення логічного типу, що вказує чи входить елемент в множину.



Перевірка на входження | Перетин | Об’єднання

Симетрична різниця | Різниця

Рисунок – Подання операцій над множинами діаграмами Ейлєра

№16

Структури

На відміну від масивів чи множин, усі елементи яких однотипні, структура може містити елементи різних типів.

Елементи структури називаються полями структури і можуть мати довільний тип, крім типу цієї ж структури, але можуть бути покажчиками на неї. Якщо при описі структури відсутній тип структури, обов'язково повинен бути вказаний список змінних, покажчиків або масивів визначеної структури.

Звернення до окремих полів структури замінюються на їхні адреси ще на етапі компіляції.

Найважливішою операцією для структури є операція доступу до вибраного поля структури – операція кваліфікації.

Над вибраним полем структури можливі будь-які операції, які допустимі для типів цього поля.

Більшість мов програмування підтримує деякі операції, які працюють із структурою, як з єдиним цілим, а не з окремими її полями. Це операція присвоєння значення одного запису іншому однотипному запису, при цьому відбувається по елементне копіювання.

Об'єднання

Об'єднання представляють собою частковий випадок структури, усі поля якої розміщуються за однією ж і тою ж адресою. Формат опису такий же, як і в структури. Довжина об'єднання рівна найбільшій із довжин його полів. У кожен момент часу в змінній типу об'єднання зберігається тільки одне значення, і відповідальність за його правильне використання лягає на програміста.

Об'єднання застосовуються для економії пам'яті в тих випадках, коли відомо, що більше одного поля одночасно не потрібно, а також для різної інтерпретації одного і того ж бітового представлення.

Дуже часто деякі об'єкти програми відносяться до одного й того ж класу, відрізняючись лише деякими деталями. У цьому випадку застосовують

комбінацію структурного типу і об’єднання. Об’єднання використовують як поля структури, при цьому в структурі включають поле, яке визначає, який саме елемент об’єднання використовується в кожний момент.

У загальному випадку змінна структура буде складатися з трьох частин: набір спільних компонентів, мітки активного компоненту і частини зі змінними компонентами.

Бітові типи

В ряді задач може стати в нагоді робота з окремими бінарними розрядами даних. Частіше всього такі задачі виникають в системному програмуванні, коли, наприклад, окремий розряд зв’язаний з станом окремого апаратного перемикача або окремої шини передачі даних. Дані такого типу представляються у вигляді набору бітів, які упаковані в байти або слова, і логічно не зв’язаних один з одним. Операції над такими даними забезпечують доступ до вибраного біта даного.

Бітові поля – це особливий вид полів структури. Вони використовуються для компактного розміщення даних, наприклад, прапорців типу „так/ні”. Мінімально адресована комірка пам’яті – 1 байт, а для зберігання прапорця достатньо одного біта.

Бітові поля описуються за допомогою структурного типу. Бітові поля можуть бути довільного цілого типу. Ім’я поля може бути відсутнім, такі поля використовуються для вирівнювання на апаратну межу. Доступ до поля здійснюється звичайним способом – за іменем.

Над бітовими типами можливі три групи специфічних операцій: операції алгебри логіки, операції зсуву, операції порівняння. Операції бульової алгебри – НІ („~”), АБО („|”), І („&”), виключне АБО („^”). Ці операції і за назвою, і за змістом подібні на операції над логічними аргументами, але відмінність у їх застосуванні до бітових аргументів полягає в тому, що операції виконуються над окремими розрядами. В мові C/C++ для побітових і загальних логічних операцій використовуються різні позначення.

Операції зсуву виконують зміщення бінарного коду на задану кількість розрядів ліворуч або праворуч. Із трьох можливих типів зсуву (арифметичний,

логічний, циклічний) в мовах програмування частіше реалізується лише логічний.

Таблиці

Елементами векторів і масивів можуть бути інтегровані структури. Одна з таких складних структур – таблиця. З фізичної точки зору таблиця є вектором, елементами якого є структури. Характерною логічною особливістю таблиць є те, що доступ до елементів таблиці проводиться не за номером (індексом), а за ключем – значення однієї з властивостей об'єкту, який описується структурою-елементом таблиці. Ключ – це властивість, що ідентифікує дану структуру в множині однотипних структур і є, як правило, унікальним в даній таблиці. Ключ може включатися до складу структури і бути одним з його полів, але може і не включатися в структуру, а обчислюватися за деякими її властивостями. Таблиця може мати один або декілька ключів.

Основною операцією при роботі з таблицями є операція доступу до структури за ключем. Вона реалізовується процедурою пошуку. Оскільки пошук може бути значне більш ефективним в таблицях, впорядкованих за значеннями ключів, досить часто над таблицями необхідно виконувати операції сортування.

Іноді розрізняють таблиці з фіксованою і із змінною довжиною структури. Очевидно, що таблиці, які об'єднують структури ідентичних типів, будуть мати фіксовані довжини структур. Необхідність в змінній довжині може виникнути в задачах, подібних до тих, які розглядалися для об'єднань. Як правило таблиці для таких задач і складаються із структур до складу яких входять об'єднання, тобто зводяться до фіксованої (максимальній) довжини структури. Значно рідше зустрічаються таблиці з дійсно змінною довжиною структури. Хоча в таких таблицях і економиться пам'ять, але можливості роботи з такими таблицями обмежені, оскільки за номером структури неможливо визначити її адресу. Таблиці із структурами змінної довжини обробляються тільки послідовно – в порядку зростання номерів структур. Доступ до елемента такої таблиці звичайно здійснюється в два кроки. На першому кроці вибирається постійна частина структури, в якій міститься, – в явному чи неявному вигляді – довжина структури. На другому кроці

вибирається змінна частина структури у відповідності з її довжиною. Додавши до адреси поточної структури її довжину, одержують адресу наступної структури.

НАПІВСТАТИЧНІ СТРУКТУРИ ДАНИХ

№17

Характерні особливості напівстатичних структур

Напівстатичні структури даних характеризуються наступними ознаками:

- вони мають змінну довжину і прості процедури її зміни;
- зміна довжини структури відбувається в певних межах, не перевищуючи якогось максимального (граничного) значення.

Якщо напівстатичну структуру розглядати на логічному рівні, то це послідовність даних, зв’язана відносинами лінійного списку. Доступ до елемента може здійснюватися за його порядковим номером.

Фізичне представлення напівстатичних структур даних в пам’яті – це звичайно послідовність комірок в пам’яті, де кожний наступний елемент розташований в пам’яті в наступній комірці.

Фізичне представлення може мати також вид одно-направленого зв’язного списку (ланцюжки), де кожний наступний елемент адресується покажчиком, який знаходиться в поточному елементі. У цьому випадку обмеження на довжину структури менш строгі.

№18

Стеки

Стеком називається множина деякої змінної кількості даних, над якою виконуються наступні операції:

- Поповнення стеку новими даними;
- Перевірка, яка визначає чи стек пустий;
- Перегляд останніх добавлених даних;

• Знищення останніх добавлених даних.

На основі такого функціонального опису, можна сформувати логічний опис. Стек – це такий послідовний список із змінній довжиною, включення і виключення елементів з якого виконуються тільки з одного боку списку.

Застосовуються і інші назви стеку – магазин, пам’ять що функціонує за принципом LIFO (Last – In – First – Out – „останнім прийшов – першим вийшов”). Самий „верхній” елемент стеку, тобто останній добавлений і ще не знищений, відіграє особливу роль: саме його можна модифікувати й знищувати. Цей елемент називають вершиною стеку. Іншу частину стеку називають тілом стеку. Тіло стеку, само собою, є стеком: якщо виключити зі стеку його вершину, то тіло перетворюється в стек.

Основні операції над стеком – включення нового елемента (push – заштовхувати) і виключення елемента зі стеку (pop – вискакувати).

Корисними можуть бути також допоміжні операції: • визначення поточної кількості елементів в стеку;

• очищення стеку;

• „неруйнуюче” читання елемента з вершини стека, яке може бути реалізоване, як комбінація основних операцій – виключити елемент зі стеку та включити його знову в стек.

При представленні стеку в статичній пам’яті для стеку виділяється пам’ять, як для вектора. В описі цього вектора окрім звичайних для вектора параметрів повинен знаходитися також покажчик стеку – адреса вершини стека. Обмеження даного представлення полягає в тому, що розмір стеку обмежений розмірами вектора.

Покажчик стеку може вказувати або на перший вільний елемент стеку, або на останній записаний в стек елемент. Однаково, який з цих двох варіантів вибрати, важливо надалі строго дотримуватися його при обробці стеку. При занесенні елементу в стек елемент записується на місце, яке визначається покажчиком стеку, потім покажчик модифікується так, щоб він вказував на наступний вільний елемент (якщо покажчик вказує на останній записаний елемент, то спочатку модифікується покажчик, а потім проводиться запис

елемента). Модифікація покажчика полягає в надбавці до нього або у відніманні від нього одиниці (стек росте у бік збільшення адреси).

Операція виключення елемента полягає в модифікації покажчика стеку (в напрямку, зворотному модифікації при включенні) і вибірці значення, на яке вказує покажчик стеку. Після вибірки комірка, в якій розміщувався вибраний елемент, вважається вільною.

Операція очищення стеку зводиться до запису в покажчик стеку початкового значення – адреси початку виділеної ділянки пам’яті.

Визначення розміру стека зводиться до обчислення різниці покажчиків: покажчика стеку й адреси початку ділянки.

При зв’язному представленні стеку кожен елемент стеку складається із значення і покажчика, який вказує на попередньо занесений у стек елемент. Зв’язне представлення викликає втрату пам’яті, що викликане наявністю покажчика в кожному елементі стеку, і представляє інтерес тільки у випадку, коли важко визначити максимальний розмір стеку.

Отже, для зв’язного представлення стеку потрібно, щоб кожен його елемент описувався структурою, яка поєднує дані і покажчик на наступний елемент.

Для виконання операцій над стеком потрібен один покажчик на вершину стеку. Створення пустого стеку полягатиме у присвоєнні покажчику на вершину нульового значення, що означатиме, що стек пустий.

№19

Послідовність кроків для добавлення елементу в стек складається з декількох кроків:

1. Виділити пам’ять під новий елемент стеку;

2. Занесення значення в інформаційне поле;

3. Встановлення зв’язку між ним і „старою” вершиною стеку;

4. Переміщення вершини стеку на новий елемент.

Вилучення елементу зі стеку також проводять за кілька кроків:

1. Зчитування інформації з інформаційного поля вершини стеку;

2. Встановлення на вершину стеку допоміжного покажчика;

3. Переміщення покажчика вершини стеку на наступний елемент;

4. Звільнення пам’яті, яку займає „стара” вершина стеку.

№ 20

Черга

Чергою називається множина змінної кількості даних, над якою можна виконувати наступні операції:

- Поповнення черги новими даними;
- Перевірка, яка визначає чи пуста черга;
- Перегляд перших добавлених даних;
- Знищення самих перших добавлених даних.

На основі такого функціонального опису, можна сформувати логічний опис. Чергою FIFO (First – In – First – Out – „першим прийшов – першим вийшов”). називається такий послідовний список із змінній довжиною, в якому включення елементів виконується тільки з одного боку списку (хвіст черги), а виключення – з другого боку (голова черги).

Основні операції над чергою – ті ж, що і над стеком – включення, виключення, визначення розміру, очищення, „неруйнуюче” читання.

При представленні черги вектором в статичній пам’яті на додаток до звичайних для опису вектора параметрів в ньому повинні знаходитися два покажчики: на голову і на хвіст черги. При включенні елемента в чергу елемент записується за адресою, яка визначається покажчиком на хвіст, після чого цей покажчик збільшується на одиницю. При виключенні елемента з черги вибирається елемент, що адресується покажчиком на голову, після чого цей покажчик зменшується на одиницю.

Очевидно, що з часом покажчик на хвіст при черговому включенні елемента досягне верхньої межі тієї ділянки пам’яті, яка виділена для черги. Проте, якщо операції включення чергувати з операціями виключення

елементів, то в початковій частині відведеної під чергу пам’яті є вільне місце. Для того, щоб місця, займані виключеними елементами, могли бути повторно використані, черга замикається в кільце: покажчики (на початок і на кінець), досягнувши кінця виділеної області пам’яті, перемикаються на її початок. Така організація черги в пам’яті називається кільцевою чергою. Можливі, звичайно, і інші варіанти організації: наприклад, всякий раз, коли покажчик кінця досягне верхньої межі пам’яті, зсовувати всі не порожні елементи черги до початку ділянки пам’яті, але як цей, так і інші варіанти вимагають переміщення в пам’яті елементів черги і менш ефективні, ніж кільцева черга.

У початковому стані покажчики на голову і хвіст вказують на початок ділянки пам’яті. Рівність цих двох покажчиків є ознакою порожньої черги. Якщо в процесі роботи з кільцевою чергою кількість операцій включення перевищує кількість операцій виключення, то може виникнути ситуація, в якій покажчик кінця „наздожене” покажчик початку. Це ситуація заповненої черги, але якщо в цій ситуації покажчики порівняються, ця ситуація буде така ж як при порожній черзі. Для розрізнення цих двох ситуацій до кільцевої черги пред’являється вимога, щоб між покажчиком кінця і покажчиком початку залишався „проміжок” з вільних елементів. Коли цей „проміжок” скорочується до одного елемента, черга вважається заповненою і подальші спроби запису в неї блокуються. Очищення черги зводиться до запису одного і того ж (не обов’язково початкового) значення в обидва покажчики. Визначення розміру полягає в обчисленні різниці покажчиків з урахуванням кільцевої природи черги.

При зв’язному представленні черги кожен елемент черги складається із значення і покажчика, який вказує на попередньо занесений у чергу елемент.

Зв’язне представлення викликає втрату пам’яті, що викликане наявністю покажчика в кожному елементі черги, і представляє інтерес тільки у випадку, коли важко визначити максимальний розмір черги. Для зв’язного представлення черги потрібно, щоб кожен його елемент описувався структурою, яка поєднує дані і покажчик на наступний елемент.

Для виконання операцій над чергою потрібно два покажчики: на голову і хвіст черги. Створення пустої черги полягатиме у присвоєнні покажчикам на голову і хвіст черги нульових значень, що означатиме, що черга пуста.

№21

Послідовність кроків для добавлення елемента в кінець черги складається з декількох кроків:

1. Виділити пам’ять під новий елемент черги;

2. Занесення значення в інформаційне поле;

3. Занесення нульового значення в покажчик;

4. Встановлення зв’язку між ним і останнім елементом черги і новим, враховуючи випадок пустої черги;

5. Переміщення покажчика кінця черги на новий елемент.

Вилучення елементу з черги також проводять за кілька кроків:

1. Зчитування інформації з інформаційного поля голови черги

2. Встановлення на голову черги допоміжного покажчика;

3. Переміщення покажчика початку черги на наступний елемент;

4. Звільнення пам’яті, яку займав перший елемент черги.

В реальних задачах іноді виникає необхідність у формуванні черг, відмінних від приведених структур. Порядок вибірки елементів з таких черг визначається пріоритетами елементів. Пріоритет в загальному випадку може бути представлений числовим значенням, яке обчислюється або на підставі значень яких-небудь полів елемента, або на підставі зовнішніх чинників. Так попередньо наведені структури стек і черги можна трактувати як пріоритетні черги, в яких пріоритет елемента залежить від часу його включення в структуру. При вибірці елемента всякий раз вибирається елемент з щонайбільшим пріоритетом.

Черги з пріоритетами можуть бути реалізовані на лінійних структурах – в суміжному або зв’язному представленні. Можливі черги з пріоритетним включенням – в яких послідовність елементів черги весь час підтримується впорядкованою, тобто кожний новий елемент включається на те місце в послідовності, яке визначається його пріоритетом, а при виключенні завжди вибирається елемент з голови. Можливі і черги з пріоритетним виключенням

– новий елемент включається завжди в кінець черги, а при виключенні в черзі шукається (цей пошук може бути тільки лінійним) елемент з максимальним пріоритетом і після вибірки вилучається з послідовності. І в тому, і в іншому варіанті потрібний пошук, а якщо черга розміщується в статичній пам’яті – ще і переміщення елементів.

№22

Деки

Дек – особливий вид черги. Дек (deq – double ended queue, тобто черга з двома кінцями) – це такий послідовний список, в якому як включення, так і виключення елементів, може здійснюватися з будь-якого з двох кінців списку.

Так само можна сформулювати поняття деку, як стек, в якому включення і виключення елементів може здійснюватися з обох кінців.

Деки рідко зустрічаються у своєму первісному визначенні. Окремий випадок деку – дек з обмеженим входом і дек з обмеженим виходом.

Логічна і фізична структури деку аналогічні логічній і фізичній структурі кільцевої черги. Проте, стосовно деку доцільно говорити не про голову і хвіст, а про лівий і правий кінець.

Над деком доступні наступні операції:

- включення елемента праворуч;
- включення елемента ліворуч;
- виключення елемента з права;
- виключення елемента з ліва;
- визначення розміру;
- очищення.

Фізична структура деку в статичній пам’яті ідентична структурі кільцевої черги.

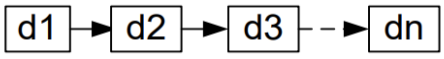
№ 23

Лінійні списки

Лінійні списки є узагальненням попередніх структур; вони дозволяють представити множину так, щоб кожний елемент був доступний і при цьому не потрібно було б зачіпати деякі інші.

Списки є досить гнучкою структурою даних, так як їх легко зробити більшими або меншими, і їх елементи доступні для вставки або вилучення в будь-якій позиції списку. Списки також можна об’єднувати або розділяти на менші списки.

Лінійний список – це скінчена послідовність однотипних елементів (вузлів), можливо, з повторенням. Кількість елементів у послідовності називається довжиною списку. Вона в процесі роботи програми може змінюватися. Лінійний список L , що складається з елементів d_1, d_2,…, d_n, які мають однаковий тип, записують у вигляді L = [d_1, d_2,…, d_n] , або зображають графічно.



Важливою властивістю лінійного списку є те, що його елементи можна лінійно впорядкувати у відповідності з їх позицією в списку.

Для формування абстрактного типу даних на основі математичного визначення списку потрібно задати множину операторів, які виконуються над об’єктами типу список. Проте не існує однієї множини операторів, які виконуються над списками, які задовольняють відразу всі можливі застосування.

Найчастіше зі списками доводиться виконувати такі операції:

- Знайти елемент із заданою властивістю;
- Визначити і-й елемент у лінійному списку;
- Внести додатковий елемент до або після вказаного вузла;
- Вилучити певний елемент списку;
- Впорядкувати вузли лінійного списку в певному порядку.

У реальних мовах програмування не існує якої-небудь структури даних для зображення лінійного списку так, щоб усі операції над ним виконувалися

в однаковій мірі ефективно. Тому при роботі з лінійними списками важливе значення має подання лінійних списків, які використовуються в програмі, таким чином, щоб була забезпечена максимальна ефективність і за часом виконання програми, і за обсягом потрібної їй пам’яті.

Лінійний список є послідовність об’єктів. Позиція елемента в списку має інший тип даних, відмінний від типу даних елемента списку, і цей тип залежить від конкретної фізичної реалізації.

№24

Над лінійним списком допустимі наступні операції.

Операція вставки – вставляє елемент в конкретну позицію в списку, переміщуючи елементи від цієї позиції і далі в наступну, більш вищу позицію.

Операція локалізації – повертає позицію об’єкта в списку. Якщо в списку об’єкт зустрічається декілька разів, то повертається позиція першого від початку списку об’єкта. Якщо об’єкта немає в списку, то повертається значення, яке рівне довжині списку, збільшене на одиницю.

Операція вибірки елемента з списку – повертає елемент, який знаходиться в конкретній позиції списку. Результат не визначений, якщо в списку немає такої позиції.

Операція вилучення – вилучає елемент в конкретній позиції зі списку. Результат невизначений, якщо в списку немає вказаної позиції.

Операції вибірки попереднього і наступного елемента – повертають відповідно наступній і попередній елемент списку відносно конкретної позиці в списку.

Функція очистки списку робить список пустим.

Основні методи зберігання лінійних списків поділяються на методи послідовного і зв’язного зберігання. При виборі способу зберігання в конкретній програмі слід враховувати, які операції і з якою частотою будуть виконуватися над лінійними списками, вартість їх виконання та обсяг потрібної пам’яті для зберігання списку.

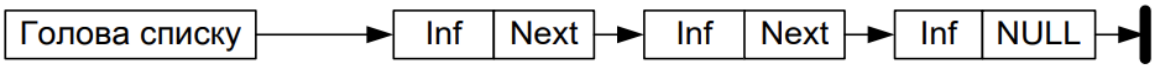
Найпростіша форма представлення лінійного списку — це вектор. Визначивши таким чином список можна по чергово звертатися до них в циклі і виконувати необхідні дії. Однак при такому представленні лінійного списку не вдасться уникнути фізичного переміщення елементів, якщо потрібно добавляти нові елементи, або вилучати існуючі. Набагато швидше вилучати елементи можна за допомогою простої схеми чистки пам'яті. Замість вилучення елементів із списку, їх помічають як невикористані.

Більш складною організацією при роботі зі списками є розміщення в масиві декількох списків або розміщення списку без прив'язки його початку до першого елемента масиву.

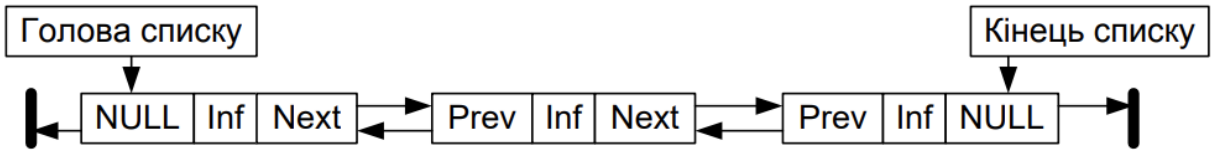
№25

При зв'язному представленні лінійного списку кожен його елемент складається із значення і покажчика, який вказує на наступний елемент у списку.

На наступному рисунку приведена структура однозв'язного списку. Кожний список повинен мати особливий елемент, який називається покажчиком на початок списку, або головою списку.

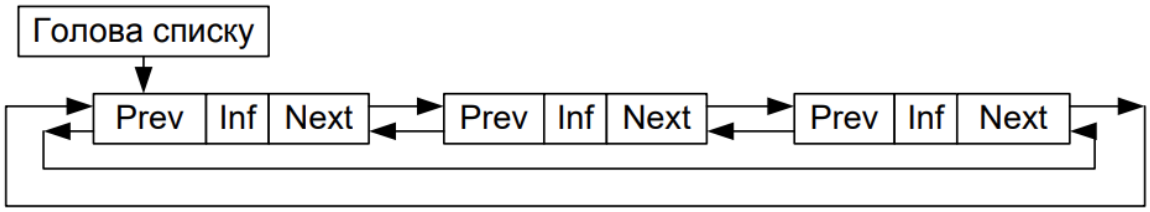


Проте, обробка однозв'язного списку не завжди зручна, оскільки відсутня можливість просування в протилежну сторону. Таку можливість забезпечує двохзв'язний список, кожний елемент якого містить два покажчики: на наступний і попередній елементи списку.



Для зручності обробки списку додають ще один особливий елемент – покажчик кінця списку. Наявність двох покажчиків в кожному елементі ускладнює список і приводить до додаткових витрат пам'яті, але в той же час забезпечує більш ефективне виконання деяких операцій над списком.

Різновидом розглянутих видів лінійних списків є кільцевий список, який може бути організований на основі як однозв'язного, так і двох-зв'язного списків. При цьому в однозв'язному списку покажчик останнього елемента повинен вказувати на перший елемент; в двох-зв'язному списку в першому і останньому елементах відповідні покажчики змінюються.



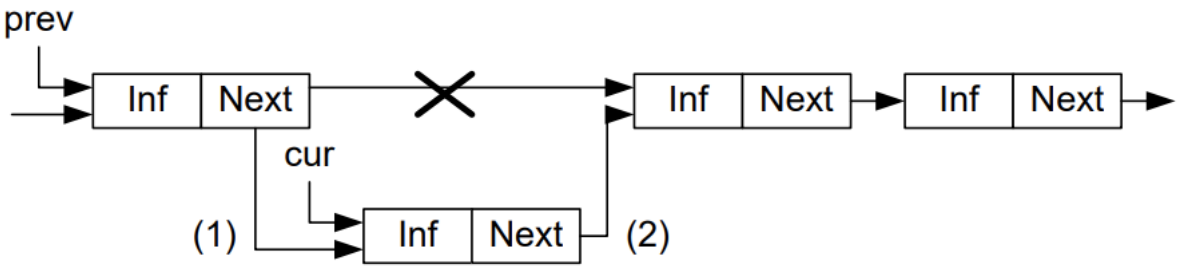
При роботі з такими списками дещо спрощуються деякі процедури, проте, при перегляді такого списку слід приймати деякі запобіжні засоби, щоб не потрапити в нескінченний цикл.

В пам'яті список є сукупністю опису однакових за розміром і форматом структур, які розміщені довільно в деякій ділянці пам'яті і пов'язані одна з одною в лінійно впорядкований ланцюжок за допомогою покажчиків. Структура містить інформаційні поля і поля покажчиків на сусідні елементи списку, причому деякими полями інформаційної частини можуть бути покажчики на блоки пам'яті з додатковою інформацією, що відноситься до елемента списку.

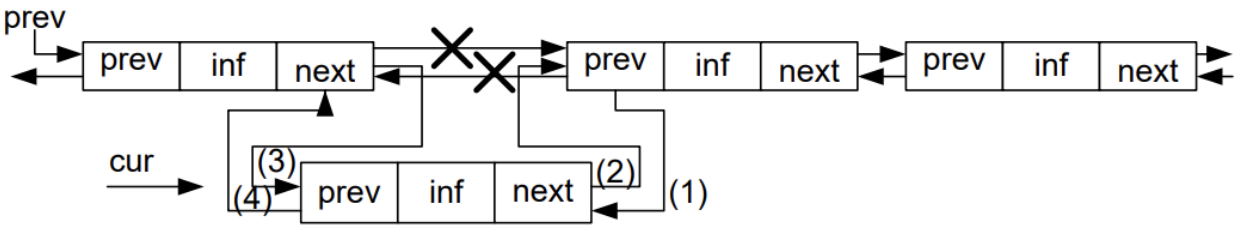
№26

Розглянемо деякі прості операції над лінійними списками.

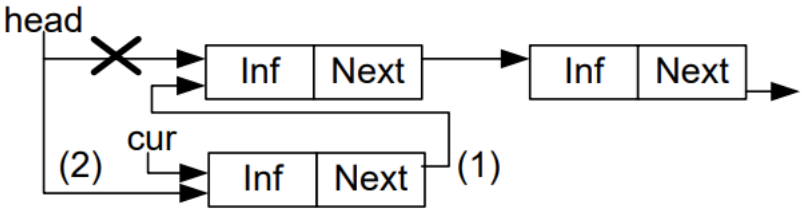
Вставка елемента в середину однозв'язного списку:



Вставка елемента в двох-зв'язний список:

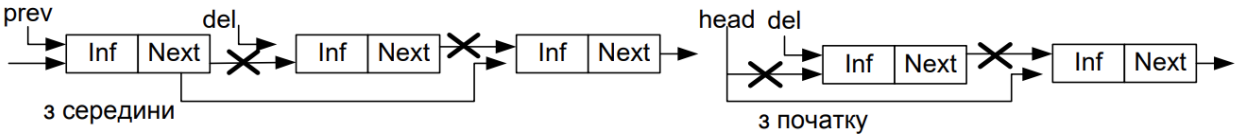


Наведені приклади забезпечують вставку в середину списку, але не можуть бути застосовані для вставки на початок списку. При такій операції повинен модифікуватися покажчик на початок списку:

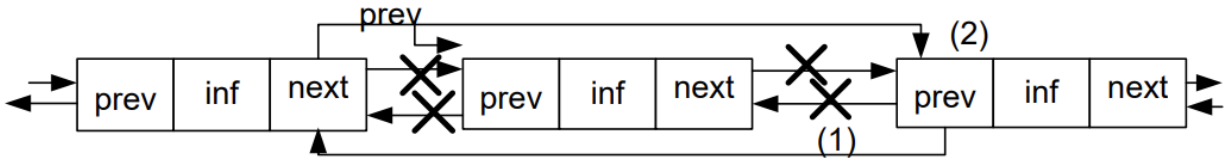


№27

Видалення елемента з однозв’язного списку для двох варіантів – з середини і з голови:



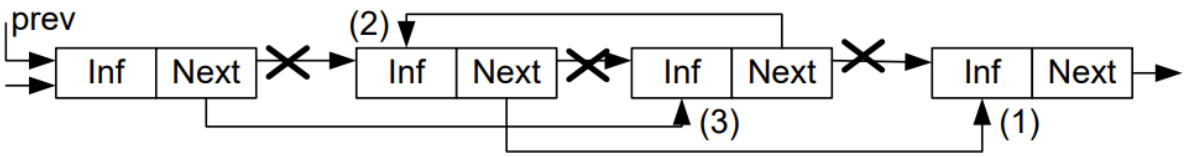
Видалення елемента з двох-зв’язного списку вимагає корекції більшої кількості покажчиків:



Процедура видалення елемента з двох-зв’язного списку виявиться навіть простішою, ніж для однозв’язного, оскільки в ній не потрібно шукати попередній елемент, він вибирається за покажчиком назад.

Змінність динамічних структур даних допускає не тільки зміни розміру структури, але і зміни зв’язків між елементами. Для зв’язних структур зміна зв’язків не вимагає пересилки даних в пам’яті, а тільки зміни покажчиків в

елементах зв’язної структури. В якості прикладу приведена перестановка двох сусідніх елементів списку. В алгоритмі перестановки в однозв’язному списку виходили з того, що відома адреса елемента, який передує парі, в якій проводиться перестановка. В приведеному алгоритмі також не враховується випадок перестановки початкових елементів списку.



У процедурі перестановки для двох-зв’язного списку неважко врахувати і перестановку на початку списку.

№28

Мультисписки

В програмних системах, які обробляють об’єкти складної структури, можуть вирішуватися різні підзадачі, кожна з яких вимагає обробки можливо не всієї множини об’єктів, а лише якоїсь його підмножини.

Для того, щоб при вибірці кожної підмножини не виконувати повний перегляд з відсіванням записів, які до необхідної підмножини не відносяться, в кожний запис включаються додаткові поля посилань, кожне з яких зв’язує в лінійний список елементи відповідної підмножини. В результаті виходить багато-зв’язковий список або мультисписок, кожний елемент якого може входити одночасно в декілька однозв’язних списків.

До переваг мультисписків крім економії пам’яті (при множині списків інформаційна частина існує в єдиному екземплярі) слід віднести також цілісність даних – в тому сенсі, що всі підзадачі працюють з однією і тією ж версією інформаційної частини і зміни в даних, зроблені одній підзадачей негайно стають доступними для іншої підзадачі.

Кожна підзадача працює з своєю підмножиною як з лінійним списком, використовуючи для цього певне поле зв’язку. Специфіка мультисписку виявляється тільки в операції виключення елемента із списку. Виключення елемента з якого-небудь одного списку ще не означає необхідності видалення елемента з пам’яті, оскільки елемент може залишатися у складі інших списків.

Пам’ять повинна звільнятися тільки у тому випадку, коли елемент вже не входить ні в один з приватних списків мультисписку. Звичайно задача видалення спрощується тим, що один з приватних списків є головним – в нього обов’язково входять всі наявні елементи. Тоді виключення елемента з будь-якого неголовного списку полягає тільки в зміні покажчиків, але не в звільненні пам’яті. Виключення ж з головного списку вимагає не тільки звільнення пам’яті, але і зміні покажчиків як в головному списку, так і у всіх неголовних списках, в які елемент, що видаляється, входив.

№29

Стрічки

Стрічка – це лінійно впорядкована послідовність символів, які належать до скінченої множини символів, яка називається алфавітом.

Стрічки мають наступні важливі властивості:

- їхня довжина, як правило, змінна, хоч алфавіт фіксований;
- звичайне звернення до символів стрічки йде з будь-якого одного боку послідовності (важлива впорядкованість послідовності, а не її індексація);
- метою доступу до стрічки є на окремий її елемент, а ланцюжок символів.

Кажучи про стрічки, звичайно мають на увазі текстові стрічки – стрічки, що складаються з символів, які входять в алфавіт якої-небудь вибраної мови, цифр, розділових знаків і інших службових символів. Текстова стрічка є найбільш універсальною формою представлення будь-якої інформації.

Хоча стрічки й розглядаються в частині, яка присвячена напівстатичним структурам даних, в тих або інших конкретних задачах змінність стрічок може варіюватися від повної її відсутності до практично необмежених можливостей зміни. Орієнтація на ту чи іншу міру мінливості стрічок визначає і фізичне представлення їх в пам’яті і особливості виконання операцій над ними. В більшості мов програмування стрічки представляються саме як напівстатичні структури.

Базовими операціями над стрічками є:

- визначення довжини стрічки;
- присвоєння стрічки;
- конкатенація (зчеплення) стрічок;
- виділення підстрічки;
- пошук входження.

Операція визначення довжини стрічки має вид функції, яка повертає значення – ціле число – поточна кількість символів в стрічці. Операція присвоєння має такий же сенс, що і для інших типів даних.

Операція порівняння стрічок має такий же сенс, що і для інших типів даних. Порівняння стрічок проводиться за наступними правилами. Порівнюються перші символи двох стрічок. Якщо символи не рівні, то стрічка, що містить символ, місце якого в алфавіті ближче до початку, вважається меншою. Якщо символи рівні, порівнюються другі, треті і т.д. символи. При досягненні кінця в одній з стрічок стрічка меншої довжини вважається меншою. При рівності довжин стрічок і попарній рівності всіх символів в них стрічки вважаються рівними.

Результатом операції зчеплення двох стрічок є стрічка, довжина якої рівна сумарній довжині стрічок-операндів, а значення відповідає значенню першого операнда, за яким безпосередньо слідує значення другого операнда.

Операція виділення підстрічки виділяє з початкової стрічки послідовність символів, починаючи із заданої позиції, із заданою довжиною.

Операція пошуку входження знаходить місце першого входження підстрічки еталону в початкову стрічку. Результатом операції може бути номер позиції в початковій стрічці, з яким починається входження еталону або покажчик на початок входження. У разі відсутності входження результатом операції повинне бути деяке спеціальне значення, наприклад, від’ємний номер позиції або порожній покажчик.

Найпростішим способом є представлення стрічки у вигляді вектора постійної довжини. При цьому в пам’яті відводиться фіксована кількість байт, в які записуються символи стрічки. Якщо стрічка менша відведеного під неї

вектора, то зайві місця заповнюються пропусками, а якщо стрічка виходить за межі вектора, то зайві (праві) символи повинні бути відкинуті.

Можливе представлення стрічки вектором змінної довжини з ознакою завершення. Цей і всі подальші за ним методи враховують змінну довжину стрічок. Ознака завершення – це особливий символ, який належить до алфавіту (таким чином, корисний алфавіт виявляється меншим на один символ), і займає ту ж кількість розрядів, що і всі інші символи. Витрати пам'яті при цьому способі складають 1 символ на рядок.

Окрім ознаки завершення можна використати лічильник символів – це ціле число, і для нього відводиться достатня кількість бітів, щоб їх з надлишком вистачало для представлення довжини найдовшої стрічки, яку можна представити. При використовуванні лічильника символів можливий довільний доступ до символів в межах стрічки.

Представлення стрічок списком у пам'яті забезпечує гнучкість у виконанні різноманітних операцій над ними (зокрема, операцій включення і виключення окремих символів і цілих ланцюжків) і використовування системних засобів управління пам'яттю при виділенні необхідного об'єму пам'яті для стрічки. Проте, при цьому виникають додаткові затрати пам'яті. Іншим недоліком такого представлення стрічок є те, що логічно сусідні елементи стрічки не є фізично сусідніми в пам'яті. Це ускладнює доступ до груп елементів стрічки в порівнянні з доступом у векторному представленні.

При представленні стрічки однозв'язним лінійним списком кожний символ стрічки представляється у вигляді елемента зв'язного списку; елемент містить код символу і покажчик на наступний елемент. Одностороннє зчеплення представляє доступ тільки в одному напрямі уздовж стрічки. При використанні двох-зв'язних лінійних списків у кожний елемент списку додається також покажчик на попередній елемент. Двостороннє зчеплення допускає двосторонній рух уздовж списку, що може значно підвищити ефективність виконання деяких стрічкових операцій.

Блочно-зв'язне представлення стрічок дозволяє в більшості операцій уникнути витрат, які пов'язані з управлінням динамічною пам'яттю, але в той

же час забезпечує достатньо ефективне використовування пам’яті при роботі з стрічками змінної довжини.

ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

№30

Зв'язне представлення даних в пам'яті

Динамічні структури за визначенням характеризуються відсутністю фізичної суміжності елементів структури в пам’яті, непостійністю і непередбачуваністю розміру (кількість елементів) структури в процесі її обробки.

Оскільки елементи динамічної структури розташовуються за не передбачуваними адресами пам’яті, адресу елемента такої структури не можна обчислити за адресою початкового або попереднього елемента. Для встановлення зв’язку між елементами динамічної структури використовуються покажчики, через які встановлюються явні зв’язки між елементами. Таке представлення даних в пам’яті називається зв’язним.

Елемент динамічної структури складається з двох полів:

- інформаційного поля або поля даних, в якому містяться ті дані, заради яких і створюється структура;
- поле зв’язку, в якому міститься один або декілька покажчиків, які зв’язують даний елемент з іншими елементами структури.

Коли зв’язне представлення даних використовується для вирішення прикладної задачі, для кінцевого користувача „видимим” робиться тільки вміст інформаційного поля, а поле зв’язку використовується тільки програмістом-розробником.

№31

Переваги зв’язного представлення даних:

- можливість забезпечення значної змінності структур;
- розмір структури обмежується тільки доступним об’ємом машинної пам’яті;

• при зміні логічної послідовності елементів структури потрібно виконати не переміщення даних в пам’яті, а тільки корекцію покажчиків.

Разом з тим зв’язне представлення не позбавлене й недоліків, основні з яких:

• робота з покажчиками вимагає більш високої кваліфікації від програміста;

• на поля зв’язку витрачається додаткова пам’ять;

• доступ до елементів зв’язної структури може бути менш ефективним за часом.

Останній недолік є найбільш серйозним і саме ним обмежується застосування зв’язного представлення даних. Якщо в суміжному представленні даних для обчислення адреси будь-якого елемента у всіх випадках достатньо номера елемента і інформації, яка міститься в описі структури, то для зв’язного представлення адреса елемента не може бути обчислена з початкових даних. Опис зв’язної структури містить один або декілька покажчиків, які дозволяють увійти до структури, далі пошук необхідного елемента виконується проходженням ланцюжком покажчиків від елемента до елемента. Тому зв’язне представлення практично ніколи не застосовується в задачах, де логічна структура даних має вигляд вектора або масиву – з доступом за номером елемента, але часто застосовується в задачах, де логічна структура вимагає іншої початкової інформації доступу (таблиці, списки, дерева і т.д.).

НЕЛІНІЙНІ СТРУКТУРИ ДАНИХ

№32

Графи

Граф – це складна нелінійна багато-зв’язна динамічна структура, що відображає властивості і зв’язки складного об’єкту.

Ця багато-зв’язна структура має наступні властивості:

• на кожний елемент (вузол, вершину) може бути довільна кількість посилань;

- кожний елемент може мати зв’язок з будь-якою кількістю інших елементів;

- кожний зв’язок (ребро, дуга) може мати напрям і вагу.

У вузлах графа міститься інформація про елементи об’єкту. Зв’язки між вузлами задаються ребрами графа. Ребра графа можуть мати спрямованість, тоді вони називаються орієнтованими, в іншому випадку – неорієнтовані. Граф, усі зв’язки якого орієнтовані, називається орієнтованим графом; граф зі всіма неорієнтованими зв’язками – неорієнтованим графом; граф із зв’язками обох типів – змішаним графом.

№33

Існує два основні методи представлення графів в пам’яті комп’ютера: матричний і зв’язними нелінійними списками. Вибір методу представлення залежить від природи даних і операцій, що виконуються над ними. Якщо задача вимагає великої кількості включень і виключень вузлів, то доцільно представляти граф зв’язними списками; інакше можна застосувати і матричне представлення.

При використанні матриць суміжності їхні елементи представляються в пам’яті комп’ютера елементами масиву. При цьому, для простого графа матриця складається з нулів і одиниць, для мультиграфа – з нулів і цілих чисел, які вказують кратність відповідних ребер, для зваженого графа – з нулів і дійсних чисел, які задають вагу кожного ребра.

№34 (приклади)

Орієнтований граф представляється зв’язним нелінійним списком, якщо він часто змінюється або якщо півміри входу і виходу його вузлів великі. Багато-зв’язна структура – граф – знаходить широке застосування при організації банків даних, управлінні базами даних, в системах програмного імітаційного моделювання складних комплексів, в системах штучного інтелекту, в задачах планування і в інших сферах.

№35

Дерева

Формально дерево визначається як скінченна множина T одного або більше вузлів з наступними властивостями:

1. Існує один корінь дерева T.

2. Інші вузли (за винятком кореня) розподілені серед $M \geq 0$ непересічних множин T_1, T_2, …, T_m і кожна з множин є деревом; дерева T_1, T_2, …, T_m називаються піддеревами даного кореня T.

Дерево – це граф, який характеризується наступними властивостями:

- Існує єдиний елемент (вузол або вершина), на який не посилається ніякий інший елемент, – він називається коренем.

- Починаючи з кореня і слідуючи по певному ланцюжку покажчиків, що містяться в елементах, можна здійснити доступ до будь-якого елемента структури.

- На кожний елемент, крім кореня, є єдине посилання, тобто кожний елемент адресується єдиним покажчиком.

Назва „дерево” виникла з логічної еквівалентності деревовидної структури абстрактному дереву з теорії графів. Лінія зв’язку між парою вузлів дерева називається гілкою. Ті вузли, які не посилаються ні на які інші вузли дерева, називаються листям. Вузол, що не є листком або коренем, вважається проміжним або вузлом галуження.

В багатьох застосування відносний порядок проходження вершин на кожному окремому ярусі має певне значення. При представленні дерева в пам’яті комп’ютера такий порядок вводиться автоматично, навіть якщо він сам по собі довільний. Порядок проходження вершин на деякому ярусі можна легко ввести, позначаючи одну вершину як першу, іншу – як другу і т.д. Замість впорядковування вершин можна задавати порядок на ребрах. Якщо в орієнтованому дереві на кожному ярусі заданий порядок проходження вершин, то таке дерево називається впорядкованим деревом.