Тема 4 СТРУКТУРА ДАНИХ ПРОГРАМ

4.1. ПОНЯТТЯ СТРУКТУРИ ДАНИХ ПРОГРАМ

Під структурою даних програм у випадку розуміють безліч елементів даних, безліч зв'язків з-поміж них, і навіть характер їх організованості.
Під організованістю даних розуміється продуманий пристрій із метою раціонального використання за призначенням. Приклади організованості даних: стек, організований масивом; структура даних для зберігання інформації про студентів; файл, що має організацію текстового файлу, байтна організація фізичної пам'яті машини.
М. Вірт визначив поняття програми так:

```
Алгоритми + структури даних = програми
```

Найпростіші структури даних, реалізовані мовою програмування, називають стандартними типами даних. Багато мов програмування дозволяють з урахуванням стандартних типів будувати типи даних, визначені програмістом (користувачем).
Що ж характеризує дані змістовніше, ніж значення? У 1973 р. М. Віртом було опубліковано статтю «Типи даних — це значення». З його погляду тип даних - це безліч значень. У статті йшлося також, що дані насамперед характеризуються набором операцій, які можна виконувати над цими даними, безліччю значень. Цей погляд і дав світові згодом деякі дуже корисні ідеї. Головна формула, якої стали дотримуватися:

```
ТИП ДАНИХ = БАГАТО ЗНАЧЕНЬ + НАБІР ОПЕРАЦІЙ
```

Важливо зрозуміти, що поняття даних та операцій дуже взаємопов'язані. Нехай є деяка структура даних, для якої існує операція Length, яка повертає довжину цієї структури деяких одиницях. Виникає питання: чи є десь дані, які називаються довжиною, чи ні. З змістовної точки зору це не має значення. Якщо ця операція застосовується до рядків, ознака кінця яких нуль (null terminated string), то обчислення довжини це дійсно операція, що вимагає обчислень. Якщо ця операція застосовується до рядків, перший байт яких означає довжину рядка, а далі йде сам рядок (як Turbo Pascal), то тут відбувається просто взяття даних з пам'яті, тобто довжина може бути операцією, а може бути даними, хоча це й не має значення для програміста.
Структури даних та алгоритми є основою побудови програм. Вбудовані в апаратуру комп'ютера структури даних представлені тими регістрами та словами пам'яті, де зберігаються двійкові величини. Закладені в конструкцію апаратури алгоритми - це втілені в електронних логічних ланцюгах жорсткі правила, за якими дані внесені в пам'ять інтерпретуються як команди, що підлягають виконанню центральним процесором.
Дані, що розглядаються у вигляді послідовності бітів або байтів, мають дуже просту організацію або, іншими словами, слабо структуровані. Для людини описувати і досліджувати складні дані в термінах послідовностей бітів або байтів дуже незручно. Завдання, які вирішуються за допомогою комп'ютера, рідко виражаються мовою бітів та байтів. Як правило, дані мають форму чисел, літер, текстів, символів та більш складних структур типу послідовностей, списків та дерев.
Мови програмування високого рівня підтримують системи формальних позначень однозначного опису як абстрактних структур даних, і алгоритмів програм. Використання мнемоніки імен констант чи

змінних полегшує роботу програмісту. Для комп'ютера всі типи даних зводяться зрештою до послідовності бітів (байтів) і мнемоніка імен йому байдужа. Компілятор пов'язує кожен ідентифікатор з певною адресою пам'яті, причому він враховує інформацію про тип кожної іменованої величини з метою перевірки сумісності типів. Людина має інтуїтивну здатність розумітися на типах даних і тих операціях, які для кожного типу справедливі. Так, наприклад, не можна витягти квадратний корінь зі слова або написати число з малої літери.
Стандартні типи даних, прийняті в мовах програмування, зазвичай включають натуральні та цілі числа, дійсні (дійсні) числа, літери, рядки тощо. Склад типів даних може різнитися в різних мовах. При виконанні програми значення змінної може змінюватися багаторазово, але її тип не змінюється ніколи. Завдяки типам компілятор може перевірити коректність операцій, що виконуються над тією чи іншою змінною. Таким чином, типи змінних багато в чому визначають структуру даних.
Програмістові, який хоче, щоб його програма мала реальне застосування в деякій прикладній галузі, не слід забувати про те, що програмування – це обробка даних. Реальний програмний продукт завжди має Замовник. Замовник має вхідні дані, і він хоче, щоб за ними були отримані вихідні дані, а якими засобами це забезпечується — його зазвичай не цікавить. Таким чином, завданням створення будь-якого програмного продукту є перетворення вхідних даних у вихідні через послідовні стани проміжних даних.
Структура даних програми багато в чому визначає алгоритми. Одне й те завдання може часто вирішуватися з допомогою різних структур даних. Для вирішення однієї і тієї ж задачі, але з структурами даних, що розрізняються, зазвичай потрібні різні алгоритми. Без попередньої специфікації структури даних неможливо розпочинати складання алгоритмів.
Структура даних належить сутнісно «просторовим» поняттям: її можна звести до схеми організації інформації у пам'яті комп'ютера. Алгоритм є відповідним процедурним елементом у структурі програми — він служить рецептом розрахунку.
Перш ніж приступати до вивчення конкретних структур даних, дамо їхню загальну класифікацію за декількома ознаками.
Поняття «фізична структура даних» відбиває спосіб фізичного представлення даних у пам'яті машини і називається ще структурою зберігання, внутрішньої структурою, структурою пам'яті чи дампом.
Розгляд структури даних без урахування її представлення у машинній пам'яті називають абстрактною, чи логічною, структурою даних. У випадку між логічної і відповідної їй фізичної структурами є відмінність, внаслідок якого існують правила відображення логічної структури на фізичну структуру.
Структури даних, які застосовуються в алгоритмах, можуть бути надзвичайно складними. У результаті вибір правильного представлення даних часто служить ключем до успішного програмування і може більшою мірою позначатися на продуктивності програми, ніж деталі алгоритму.
Більшість авторів публікацій, присвячених структурам та організації даних, наголошують на тому, що знання структур даних дозволяє організувати їх зберігання та обробку максимально ефективним чином — з точки зору мінімізації витрат як пам'яті, так і процесорного часу.
Іншим не менше, а можливо, і більш важливою перевагою, що забезпечується структурним підходом до даних, є можливість структурування складної програми для досягнення її зрозумілості людині, що скорочує кількість помилок при початковому кодуванні і необхідно при подальшому супроводі.
Іншим надзвичайно продуктивним технологічним прийомом, пов'язаним із структуризацією даних, є інкапсуляція, сенс якої полягає в тому, що сконструйований новий тип даних оформляється таким чином, що його внутрішня структура стає недоступною для програміста користувача цього типу даних. Програміст, який використовує такий тип даних у своїй програмі, може оперувати даними лише через дзвінки процедур.
Навряд чи з'явиться загальна теорія вибору структур даних. Найкраще, що можна зробити, це розібратися у всіх базових «цеглинках» і зібраних із них структурах. Здатність докласти ці знання до конструювання великих систем - це справа інженерної майстерності та практики.

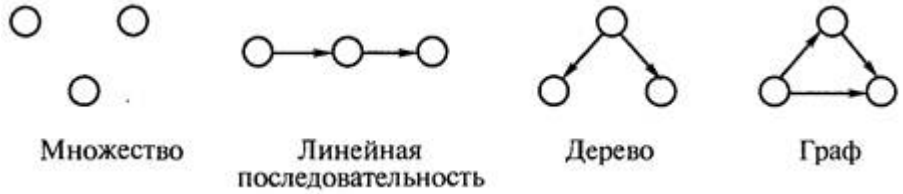
4.2. ОПЕРАЦІЇ НАД СТРУКТУРАМИ ДАНИХ

Над усіма структурами даних може виконуватися п'ять операцій: створення, знищення, вибір (доступ), оновлення, копіювання.

Операція створення полягає у виділенні пам'яті для структури даних. Пам'ять може виділятися у процесі виконання програми за першої появі імені змінної у вихідній програмі чи етапі компіляції. У ряді мов (наприклад, С) для структурованих даних, що конструюються програмістом, операція створення включає в себе також установку початкових значень параметрів, створюваної структури.
Операція знищення структур даних протилежна за своєю дією операції створення. Не всі мови дозволяють програмісту знищувати створені структури даних. Операція знищення допомагає ефективно використати оперативну пам'ять.
Операція вибору використовується програмістами для доступу до даних усередині самої структури. Форма операції доступу залежить від типу структури даних, до якої здійснюється звернення. Під час операцій вибору використовуються покажчики. У широкому значенні слова покажчик - це змінна, що визначає місце конкретної інформації в структурі даних, наприклад, змінна, що містить значення індексу статичного масиву. У вузькому значенні слова покажчик вказує на фізичну адресу чогось. В останньому випадку покажчик — це змінна, яка є носієм адреси іншої простої чи структурованої змінної, а також процедури. Ідея, що лежить в основі концепції покажчиків, полягає в тому, щоб для досягнення контролю правильності використання покажчиків пов'язати певний тип даних із конкретним покажчиком.
Операція оновлення дозволяє змінити значення даних у структурі даних. Прикладом операції оновлення є операція присвоєння чи складніша форма — передача параметрів.
Операція копіювання створює копію даних у новому місці пам'яті.
Вищезазначені п'ять операцій є обов'язковими для всіх структур і типів даних. Крім цих загальних операцій кожної структури даних можуть бути визначені операції специфічні, що працюють лише з даними зазначеного типу (даної структури).

4.3. ЗАГАЛЬНА КЛАСИФІКАЦІЯ ЛОГІЧНИХ СТРУКТУР ДАНИХ

Упорядкованість елементів структури данихє важливим її ознакою.
Програмісти можуть на власний розсуд упорядкувати дані різних програм безліччю способів. Навіть в одній і тій же структурі даних програміст може по-різному розмістити ту саму інформацію. Так, у списку студентів прізвище може передувати імені та по батькові та, навпаки, ім'я та по батькові можуть передувати прізвищу. Максимальний елемент у відсортованому масиві може бути як першим, так і останнім. Тому характер упорядкованості елементів структури, визначений програмістом, необхідно коментувати з тією чи іншою ретельністю, що визначається здоровим глуздом та мнемонікою імен. Існує безліч способів упорядкування інформації, але серед них є і загальні, найбільш часто зустрічаються і відомі більшості програмістів.
Приклад широко відомих структур даних із різною впорядкованістю наведено на рис. 4.1.
Структури за ознакою характеру упорядкованості їх елементів можна поділяти на лінійні та нелінійні. Приклади лінійних структур - масиви, рядки, стеки, черги, лінійні одно- та двозв'язкові списки. Приклади нелінійних структур багатозв'язкові списки, дерева, графи.
Прості та інтегровані структури даних. Прості – це вбудовані, стандартні, базові, примітивні структури даних, інтегровані – структуровані, похідні, композитні, складні структури даних. Інтегровані структури даних зазвичай відносять до типів даних, що визначаються програмістом.
*Прості структури*не можуть бути розчленовані на складові, більші, ніж біти і байти. З погляду фізичної структури, важливою є та обставина, що у цій машинної архітектурі, у цій системі програмування завжди можна заздалегідь сказати, який буде розмір даного простого типу і яка структура його розміщення у пам'яті. З логічного погляду прості дані є неподільними одиницями. У мовах програмування прості структури описуються простими (базовими) типами. Прості структури даних є основою для побудови складніших інтегрованих структур.
*Інтегрованими*називають такі структури даних, складовими частинами яких є інші структури даних — прості чи, своєю чергою, інтегровані. Інтегровані структури даних конструюються програмістом з допомогою засобів інтеграції даних, наданих мовами програмування.
Мінливість структур даних також є дуже важливою ознакою. Мінливість - зміна числа елементів та (або) зв'язків між елементами структури. У визначенні мінливості структури не відображено факт зміни значень елементів даних, оскільки у разі всі структури даних мали властивість мінливості. За ознакою мінливості розрізняють структури статичні та динамічні.



*Мал. 4.1.*Приклади широко відомих структур даних

Оскільки за визначенням статичні структури відрізняються відсутністю мінливості, пам'ять для них виділяється автоматично, як правило, на етапі компіляції або при виконанні в момент активізації того програмного блоку, в якому вони описані. Виділення пам'яті на етапі компіляції є настільки зручним властивістю статичних структур, що у низці завдань програмісти використовують їх навіть уявлення об'єктів, які мають мінливістю. Наприклад, коли розмір масиву невідомий заздалегідь, йому резервується максимально можливий розмір.
У ряді мов програмування поряд із статичними змінними можуть використовуватися динамічні змінні. Динамічна змінна - це як би статична змінна, але розміщується в особливій області пам'яті поза кодом програми. У будь-який момент часу пам'ять для розміщення динамічних змінних може виділятися, так і звільнятися. Слід зазначити, що пам'ять для розміщення динамічної змінної виділяється за командою програми відразу в заздалегідь зазначеному обсязі і далі не може бути змінена, тобто структури даних, побудовані на використанні динамічних змінних, мають ту ж логічну структуру і мають таку ж мінливість, як і статичні структури даних. Тому далі динамічні змінні відноситимемо до статичних структур даних.
Фізичне уявлення динамічних змінних у пам'яті — зазвичай послідовне, як і в статичних структур, розміщення значень елементів у пам'яті.
Динамічні змінні розміщуються в області пам'яті (ДРП), що динамічно розподіляється. Область ДРП знаходиться поза областю коду програми. У зарубіжних джерелах ДРП позначається терміном heap — купа. Зазвичай заповнення області ДРП здійснюється з допомогою стандартних процедур диспетчування ДРП.
Зв'язкові динамічні структури даних. *Зв'язок*- особливий продуманий логічний пристрій збереження цілісності структури даних, елементи якої можуть перебувати в довільних, несуміжних, неконтрольованих за адресацією ділянках ДРП.
Звичайно, динамічні структури даних створюються з використанням динамічних змінних, але їхній логічний пристрій такий, що до виконання процедур доступу в програмі немає змінних, значення яких відповідають значенням елементів динамічної структури.
Динамічні зв'язкові структури, чи динамічні структури, за визначенням характеризуються відсутністю фізичної суміжності елементів структури у пам'яті, мінливістю та непередбачуваністю розміру (числа елементів) структури у процесі її обробки.
Оскільки елементи зв'язкової динамічної структури розташовуються за непередбачуваними адресами пам'яті, адреса елемента такої структури може бути обчислений з адреси початкового чи попереднього елемента. Зв'язкові структури даних пов'язані у єдину сутність системою покажчиків, які у як елементах, і статичних структурах, які забезпечують доступом до особливим елементам. Такі статичні структури називають дескрипторами. Саме таке представлення даних у пам'яті називають зв'язковим.
Елемент зв'язкової динамічної структури і двох полів:

- — інформаційного поля, чи поля даних, у якому містяться ті дані (зокрема інтегровані), заради яких і створюється;
- — поля зв'язок, у кожному полі якого міститься один або кілька покажчиків, кожен із яких пов'язує даний елемент з іншими елементами структури.

Коли зв'язне уявлення даних використовується на вирішення прикладної завдання, для кінцевого користувача «видимим» робиться лише вміст інформаційного поля, а поле зв'язок використовується лише програмістом-розробником.

Переваги зв'язкового представлення даних полягають у можливості забезпечення значної мінливості структур:

- • розмір структури обмежується лише доступним об'ємом машинної пам'яті;
- • при зміні логічної послідовності елементів структури потрібно не переміщення даних у пам'яті, лише корекція покажчиків;

Недоліки зв'язного уявлення:

- • робота з вказівниками вимагає, як правило, вищої кваліфікації від програміста;
- • на поля зв'язок витрачається додаткова пам'ять;
- • доступ до елементів зв'язкової структури може бути менш ефективним за часом.

Останній недолік є найбільш серйозним і саме ним обмежується застосування зв'язкового представлення даних. Якщо суміжному поданні даних для обчислення адреси будь-якого елемента у всіх випадках достатньо було номера елемента та інформації, що міститься в дескрипторі структури, то для зв'язкового представлення адреси елемента не може бути обчислений з вихідних даних. Дескриптор зв'язкової структури містить один або кілька покажчиків, що дозволяють увійти до структури, далі пошук необхідного елемента виконується слідуванням по ланцюжку покажчиків від елемента до елемента. Тому зв'язне уявлення 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 вихідної інформації доступу (таблиці, списки, дерева тощо) .).
За ознакою фізичного розміщення структури даних розрізняють оперативні та файлові структури. Структури даних, які у оперативної пам'яті, називають оперативними структурами. Файлові структури відповідають структури даних зовнішньої пам'яті. Оперативна пам'ять є швидкою, а зовнішня – повільною.

4.4. КЛАСИФІКАЦІЯ ВИДІВ ОПЕРАТИВНИХ СТРУКТУР ДАНИХ ЗА ЇХ ЛОГІЧНИМ ПРИСТРІЙ

Часто, говорячи про ту чи іншу структуру даних, мають на увазі її логічне уявлення. Фізичне уявлення може відповідати логічному і, крім того, може істотно різнитися у різних програмних системах. Нерідко фізична структура, крім даних, містить прихований від програміста дескриптор або заголовок, що містить загальні відомості про фізичну структуру. Наявність спеціальних даних дескриптора дозволяє здійснювати контрольований щодо помилок доступом до необхідних порцій даних.
*Статичний масив*така структура даних, що характеризується:
1) фіксованим набором елементів одного й того самого типу;
2) кожен елемент має унікальний набір значень індексів;
3) кількість індексів визначає мірність масиву, наприклад, два індекси - двомірний масив, або матриця, три індекси - тривимірний масив, один індекс - одномірний масив, або вектор;
4) звернення до елементу масиву виконується на ім'я масиву та значенням індексів для даного елемента.
Статичний вектор (одномірний масив)- Структура даних з фіксованим числом елементів одного і того ж типу (частковий випадок одновимірного статичного масиву). Кожен елемент вектора має унікальний номер заданого вектора. Звертання до елемента вектора виконується на ім'я вектора та номер необхідного елемента. З використанням статичного вектора можна реалізувати стеки, черги, деки тощо.
*Статична матриця (двовимірний масив)*структура даних з фіксованим числом елементів одного й того самого типу, рівним добутку кількості стовпців та кількості рядків (частковий випадок двовимірного статичного масиву). Кожен конкретний елемент матриці характеризується одночасно значеннями двох номерів номером рядка і номером стовпця. Матриця у фізичної пам'яті — вектор. Звертання до елемента вектора виконується на ім'я матриці, номер стовпця і номер рядка, які відповідають цьому елементу.
Статичний запис -кінцеве впорядковане безліч полів, що характеризуються різним типом даних. Записи є надзвичайно зручним засобом для представлення програмних моделей реальних об'єктів

предметної області, бо, як правило, кожен такий об'єкт має набір властивостей, що характеризуються даними різних типів.
*Полем запису*може бути, у свою чергу, інтегрована структура даних вектор, масив або інший запис.
Найважливішою операцією для запису є операція доступу до обраного поля запису - кваліфікаційна операція. Практично у всіх мовах програмування позначення цієї операції має вигляд
`<ім'я змінної - записи>.<ім'я поля>`
У ряді прикладних завдань програміст може зіткнутися з групами об'єктів, набори властивостей яких перекриваються лише частково. Для завдань подібного роду розвинені мови програмування надають у розпорядження програміста записи з варіантами (union C, case Turbo Pascal).
Рядок— це лінійно впорядкована послідовність символів, що належать кінцевій множині символів, яка називається алфавітом. Говорячи про рядки, зазвичай мають на увазі текстові рядки — рядки, що складаються із символів, що входять до алфавіту
будь-якої обраної мови, цифр, розділових знаків та інших службових символів.
Базовими операціями над рядками є:
• визначення довжини рядка;
• привласнення рядків;
• конкатенація (зчеплення) рядків;
• виділення підрядка;
• пошук входження.
Операція визначення довжини рядка має вигляд функції, що повертається значення якої є цілим числом, що дорівнює поточному числу символів у рядку.
Операція присвоювання має той самий сенс, як і інших типів даних.
Порівняння рядків провадиться за такими правилами: порівнюються перші символи двох рядків. Якщо символи не рівні, то рядок, що містить символ, місце якого в алфавіті ближче до початку, вважається меншим. Якщо символи дорівнюють, порівнюються другі, треті символи і т. д. При досягненні кінця одного з рядків рядок меншої довжини вважається меншим. При рівності довжин рядків, а головне за одночасної рівності всіх символів у рядках, рядки вважаються рівними.
Результатом операції зчеплення двох рядків є рядок, довжина якого дорівнює сумарній довжині рядків-операндів, а значення відповідає значенню першого операнда, за яким безпосередньо слідує значення другого. Операція зчеплення дає результат, довжина якого у випадку більше довжин операндов. Як і у всіх операціях над рядками, які можуть збільшувати довжину рядка (привласнення, зчеплення, складні операції), можливий випадок, коли довжина результату виявиться більшою, ніж відведений йому обсяг пам'яті. Ця проблема виникає лише у тих мовах, де довжина рядка обмежується.
Операція пошуку входження знаходить місце першого входження підрядка-еталона у вихідний рядок. Результатом операції може бути номер позиції у вихідному рядку, з якого починається входження еталона або покажчик початку входження. У разі відсутності входження результатом операції має бути деяке спеціальне значення, наприклад нульовий номер позиції або порожній покажчик.
На основі базових операцій можуть бути реалізовані будь-які інші, навіть складні операції над рядками. Наприклад, операція видалення рядка символів з номерами від n1 до n2 включно.
Статичний рядок(Тип String) в мові Pascal являє собою одномірний масив, в нульовому елементі якого знаходиться дескриптор з кількістю символів у рядку, а в наступних елементах - коди символів рядка. Головний недолік статичного рядка - незмінність фізичної довжини, що призводить до неефективних витрат пам'яті.
Статична таблиця з фізичної точки зору є вектор, елементами якого є записи. Раніше було зазначено, що полями запису може бути інтегровані структури даних — вектори, масиви, інші записи. Аналогічно елементами векторів і масивів можуть бути інтегровані структури. Одна з таких складних структур – таблиця. Частою, характерною логічною особливістю таблиць і те, що доступом до елементам таблиці виробляється за номером (індексу), а, по ключу — за значенням однієї з властивостей об'єкта, описуваного записом-элементом таблиці. Ключ - це властивість, що ідентифікує цей запис у безлічі однотипних записів. Як правило, до ключа пред'являється вимога унікальності даної таблиці. Ключ може включатися до складу запису і бути одним з її полів, але може і не включатися до запису, а обчислюватись за положенням запису. Таблиця може мати один чи кілька ключів. Наприклад, при інтеграції до таблиці записів про студентів вибірка може проводитись як за особистим номером студента, так і за прізвищем.

Отже, основний операцією під час роботи з таблицями є операція доступу до запису по ключу — конкретного значення поля записи. Вона реалізується процедурою пошуку. Оскільки пошук може бути значно ефективнішим у таблицях, упорядкованих за значеннями ключів, часто над таблицями необхідно виконувати операції сортування.
Найпростішим методом пошуку елемента, що знаходиться в неупорядкованому наборі даних, за значенням його ключа є послідовний перегляд кожного елемента набору, який триває доти, доки не буде знайдено бажаний елемент. Якщо циклічно переглянуто весь набір, але елемент не знайдено, значить, ключ не міститься в наборі. Даний алгоритм може виявитися ефективним лише у випадку, якщо набір елементів не надто великий. За двох-трьох елементів цикл взагалі не потрібен!
Для досягнення високої за швидкістю ефективності використовують розрізняються алгоритми сортування та пошуку для роботи з оперативними та файловими структурами. Огляд різних алгоритмів сортування та пошуку наведено в [17].
*Списком*називають упорядковане безліч, що складається з змінного числа елементів, яких застосовні операції включення, винятки. Список, що відбиває відносини сусідства між елементами, називають лінійним. Логічні списки (та їх приватні види: стеки, черги, деки) можна реалізувати статичним вектором чи вектором як динамічної змінної, але у випадках на розмір списку накладаються обмеження. Якщо обмеження на довжину списку не допускаються, то список подається у пам'яті як зв'язковий структури. Для зняття обмежень лінійні зв'язкові списки доцільно реалізовувати динамічними структурами даних. Такі списки називатимемо динамічними.
Стек- це лінійний список з однією точкою доступу до його елементів, що називається вершиною стека. Додати або видалити елементи можна лише через його вершину. Принцип роботи стека: LIFO (Last In-First Out – останнім прийшов – першим виключається).
Основні операції над стеком:
• включення нового елемента (англ. push - заштовхувати);
• виключення елемента зі скла (англ. pop - вискакувати).
Допоміжні операції:
• визначення поточного числа елементів у стеку;
• перегляд елементів стеку (наприклад, для друку);
• очищення стеку;
• неруйнівне читання елемента з вершини стека (може бути реалізоване як комбінація основних операцій: pop та push).
Черга- Це лінійна структура даних, в один кінець якої додаються елементи, а з іншого кінця вилучаються. Принцип роботи черги: FIFO (First In - First Out - першим прийшов - першим вийшов).
Грудень(Від англ. Deq - double ended queue) - особливий вид черги у вигляді послідовного списку, в якому як включення, так і виключення елементів може здійснюватися з будь-якого з двох кінців списку. Частковий випадок дека - дек з обмеженим входом та дек з обмеженим виходом.
Розгалужений список, або дерево, це список, елементами якого можуть бути теж списки.
Нехай є покажчик однією елемент даних (вузол), званий коренем даного дерева. Корінь містить покажчики на ряд вузлів, кожен із вузлів ряду може містити покажчики на підпорядковані ним вузли і т. д. Вузли, які більше не посилаються на нові вузли, називають листям. Таким чином, дерево росте від вузла-кореня до вузлів-листя, розгалужуючись у вузлах. Вузли крім службової інформації про покажчики, що зв'язують дерево, містять корисну інформацію.
*Біранрне дерево*дерево, у кожному вузлі якого відбувається розгалуження тільки на два піддерева (гілки): ліве та праве.
*Лісом*називають кінцеве безліч дерев, що не перетинаються.
Граф— складна нелінійна багатозв'язкова динамічна структура, що відображає властивості та зв'язки складного об'єкта, має наступні властивості:
• на кожен елемент (вузол, вершину) може бути довільна кількість посилань;
• кожен елемент може мати зв'язок із будь-якою кількістю інших елементів;
• кожна зв'язка (ребро, дуга) може мати спрямування та вагу.
У вузлах графа міститься інформація про елементи об'єкта. Зв'язки між вузлами задаються ребрами графа, які можуть мати спрямованість, що показується стрілками. І тут їх називають орієнтованими, а ребра без стрілок — неориентированными.

Граф, всі зв'язки якого орієнтовані, називають орієнтованим графом чи орграфом; з усіма неорієнтованими зв'язками - неорієнтованим графом; зі зв'язками обох типів – змішаним графом. Конкретні організації структур даних та алгоритми реалізації операцій з ними розглянуті у [21, 23, 25].

4.5. ПРОЕКТУВАННЯ ТА ДОКУМЕНТУВАННЯ ОПЕРАТИВНИХ СТРУКТУР ДАНИХ

Ряд розглянутих структур даних можна реалізувати з використанням статичних структур даних, динамічних змінних та динамічних структур даних. Багатоваріантність реалізації структур вимагає ухвалення конкретного проектного рішення про спосіб їх реалізації. При прийнятті проектного рішення застосовують такі критерії, як обсяг пам'яті, можливий набір операцій, швидкість виконання операцій. Однак тривале збереження інформації можливе лише у зовнішній пам'яті, тому проектування оперативних структур даних програми має вестися у невідривному зв'язку (паралельно) із проектуванням структури файлів програми. Дані багатьох оперативних структур повинні зберігатися у файлах та відновлюватися за інформацією, записаною раніше у файл.
Нехай потрібно спроектувати програму електронної таблиці. Такий проект виконала фірма Borland Inc, коли їй знадобилася демонстраційна програма. Обґрунтування потреби та мети розробки цього проекту було розглянуто в гол. 2.
Що бачить користувач під час роботи з електронною таблицею? - Величезний двомірний масив клітин. Що користувач може записати у клітини? — Числові значення, рядки текстів та формули. Кожна клітина також повинна зберігати інформацію про формат виведення числових значень (формати: цілий, грошовий, науковий тощо). Отже, кожна клітина повинна містити атрибут того, що знаходиться в клітині: порожня клітина, числове значення у клітині, рядок тексту, коректна формула, некоректна формула. Нехай інформація про значення числа має тип розширений, речовий (10 байт); рядки тексту містять до 79 символів; інформація формули складається з поля зі значенням, розрахованого за формулою (10 байт), і навіть поля тексту формули (79 байт). Найдовша інформація у клітини з формулою: інформація формату (2 байти), значення, розраховане за формулою (10 байт), поле тексту формули (79 байт). Разом довжина інформації клітини становить 91 байт.
Нехай програма працюватиме з електронною таблицею розміром 100×100 клітин. Тоді інформація електронної таблиці у разі використання структури даних у вигляді статичної матриці займає 91×100×100 байт = 910 000 байт ≈ 889 кбайт.
Необхідний обсяг розміщення структури перевищує стандартну пам'ять комп'ютера класу IBM PC XT — 640 кбайт, тому треба відмовитися від використання структури даних як статичної матриці.
Провівши додатковий аналіз, з'ясовуємо, що з роботі з електронною таблицею більшість клітин залишається порожніми, т. е. електронна таблиця близька до розрідженої матриці. Що відомо про розріджені матриці?
Насправді (наприклад, під час роботи з кінцевими графами) зустрічаються масиви, які з певних причин можуть записуватися на згадку не повністю, а частково. Це особливо актуально для масивів настільки великих розмірів, що їх зберігання в повному обсязі пам'яті може бути недостатньо. До таких масивів відносять симетричні та розріджені масиви.
Наприклад, квадратна матриця, яка має елементи, розташовані симетрично щодо головної діагоналі, попарно рівні один одному, називають симетричною. Якщо матриця порядку n симетрична, то її фізичної структурі досить відобразити не n2, лише n(n + 1)/2 її елементів. Доступ до трикутного масиву організується таким чином, щоб можна було звертатися до будь-якого елемента вихідної логічної структури, в тому числі і до елементів, значення яких, хоч і не представлені в пам'яті, можуть бути визначені на основі значень симетричних елементів. Насправді до роботи з симетричною матрицею розробляються такі процедури:

- • формування вектора;
- • перетворення індексів матриці на індекс вектора;
- • записи у вектор елементів верхнього трикутника елементів вихідної матриці;
- • отримання значення елементів матриці з упакованого її представлення.

У цьому проектному випадку немає особливої симетрії значень елементів.

Розріджений масив - масив, більшість елементів якого рівні між собою, так що зберігати в пам'яті досить невелику кількість значень, відмінних від основного (фонового) значення інших елементів. Розрізняють два їх види:

- • масиви, в яких розташування елементів зі значеннями, відмінними від фонового значення, можуть бути описані математичними закономірностями;
- • масиви з випадковим розташуванням елементів.

У разі роботи з розрідженими масивами питання розміщення їх у пам'яті реалізуються на логічному рівні з урахуванням їхнього виду.
Пам'ятаючи про це, класифікуємо випадок електронної таблиці як структуру даних як двомірного масиву з випадковим розташуванням рідкісних елементів і натомість порожніх значень.
Звідси рішення. Скористаємося гібридною динаміко-статичною структурою зберігання клітинної інформації з використанням статичної матриці. Застосуємо статичну матрицю записів розміром кількість рядків, помножену кількість стовпців. Кожен елемент матриці складається із запису з двома полями: поля формату виведення числових значень (2 байти) та поля покажчика на інформацію клітини (4 байти).
Структура даних порожньої електронної таблиці як статичної матриці тепер займає (2 + 4) * 100 * 100 = 60 000 байт ≈ 59 кбайт. Об'єм менше 64 кбайт для єдиної статичної структури відповідає можливостям Turbo Pascal.
Процедура ініціалізації порожньої таблиці полягатиме у присвоєнні кожному полю формату значення стандартного формату та покажчика значення Nil. Обсяг пам'яті, який займає статичним масивом, під час роботи програми будь-коли змінюється.
Після закінчення введення інформації в обрану клітину, якщо клітина не порожня (значення покажчика на структуру клітини *Nil), звільняється пам'ять, виділена раніше під колишню інформацію клітини. Нова інформація клітини записується в ділянку ДРП, що дорівнює за обсягом тільки корисної інформації клітини. У відповідне поле вказівника обраної клітини записується значення вказівника виділеної ділянки ДРП. Для запису тільки корисної інформації в клітини застосовуємо записи з варіантами (union C, case Turbo Pascal).
Корисна інформація клітини включає постійне поле атрибута вмісту клітини, а також варіантні поля іншої інформації.
Нехай електронна таблиця заповнена 300 числовими значеннями, 200 текстовими рядками завдовжки 40 символів і 400 формулами з текстом формул по 30 символів. У цьому випадку для розміщення електронної таблиці в оперативній пам'яті потрібно всього
300 * (2 + 10) + 200 * (2 + 41) + 400 * (2 + 10 + 31) = 29400 байт ≈ 28,8 кбайт.
Як видно, при роботі з електронною таблицею обсяг інформації, яку займає динамічна структура клітин, зростає повільно. Остаточно приймаємо даний варіант реалізації, виділивши з атрибуту випадок помилки при розрахунку формули в окремий атрибут Error.
Нижче наведено приклад реалізації мовою Turbo Pascal структури даних електронної таблиці. Почнемо опис структури з глобальних описів:

```
Type
Real = Extended; {Потрібен співпроцесор}
Const
{Структура даних електронної таблиці}
MAXCOLS = 100; {Розмір таблиці}
MAXROWS = 100;
MAXINPUN = 79; {Довжина рядка, що вводиться}
{Значення атрибуту виду клітини}
TXT = 0; {У клітці текст}
VALUE = 1; {У клітці значення}
FORMULA = 2; {У клітці формула}
{Тип варіантної інформації клітин}
Type
TString = String [MAXINPUT]; {Тип рядків, що вводяться}
TCellRec = record {Тип інформації клітини}
Error: Boolean; {Поле помилки формули}
case Attrib: Byte of {Attrib – це поле}
```

```
TXT: (TextStr: TString); {У клітці текст}
VALUE: (Value: Real); {У клітці значення}
FORMULA: (Fvalue: Real; {У клітині формула}
Formula: TString);
end;
end;
{Тип покажчика на тип клітини}
TCellPtr = ^TCellRec;
{Тип елемента таблиці}
TCellTableElement = record
CellFormat: Word: {Формат клітини}
CellPtr: TCellPtr; {Покажчик на клітину у ДРП}
end:
{Тип масиву інформації клітин таблиці}
TCellsTable = array [1..MAXCOLS, 1..MAXROWS] of TCellPtr;
Var {Глобальні змінні}
Cells: TCellsTable; {Статична матриця всіх клітин}
CurCell: TCellPtr; {Покажчик на поточну клітинку}
CurCol, {Колонка поточної клітини}
CurRow: Word; {Рядок поточної клітини}
```

Як видно, з метою стислості викликів більшості процедур програми було ухвалено рішення про використання дуже невеликого набору глобальних змінних. При назві констант використані лише малі літери. Імена типів мають префікс "T". Імена, які часто використовуються в парі, вирівняні за довжиною, наприклад: MAXCOLS, MAXROWS, CurCol, CurRow. Два останніх імені, що використовуються парно, були вирівняні по довжині. При вирівнюванні скорочено слово column - колонка. Глобальні імена, що використовуються в багатьох процедурах, зроблені короткими.
Крім описаного у темі 1 рефакторингу імен можна проводити рефакторинг структури даних програми. При рефакторинг структури даних замість декількох самостійних масивів можливе використання таблиці і т. д. Особливу увагу при рефакторинг слід приділяти коментування логічної структури даних.

4.6. ФАЙЛОВІ СТРУКТУРИ

4.6.1. Фізична організація файлів

Файл — впорядкований набір інформації на зовнішньому носії (найчастіше дисковому носії).
Фізична інформація файлу зовнішньому носії співвідноситься з логічною структурою даних оперативної пам'яті методами доступу операційних систем.
Зазвичай файлова система операційної системи комп'ютера містить такі засоби:
• керування файлами: зберігання файлів, звернення до них, їх колективне використання та захист;
• забезпечення цілісності файлів – гарантування того, що файл містить лише те, що потрібно;
• засоби керування зовнішньою пам'яттю (розподіляють зовнішню пам'ять для розміщення файлів).
У разі диска великого обсягу на ньому може бути багато тисяч файлів. Якщо групувати всю інформацію про місцезнаходження файлів та дескриптори файлів в одному місці, пошук конкретного файлу буде займати занадто багато часу. У цьому випадку вигідно використовувати багаторівневі каталоги файлів і системне ім'я файлу формувати з ім'ям шляху від кореневої папки (кореневої директорії) до файлу (як у UNIX, MS DOS, MS Windows) або від поточної папки (поточної директорії), в якому знаходиться файл виконуваної програми.
Дескриптор файлу або блок керування файлом може містити таку інформацію:
1) рядкове ім'я файлу;
2) тип файлу (розширення імені) - інформація для користувача про передбачувану інформацію у файлі;
3) розміщення файлу у зовнішній пам'яті;
4) тип організації файлу (прямий, послідовний, індексно-послідовний тощо);
5) тип пристрою (незнімний, знімний, що допускає лише читання тощо);
6) дані (атрибути) для контролю доступу (власник, груповий користувач, допущений та загальнодоступний користувачі);
7) диспозицію (файл постійний чи тимчасовий);
8) дату та час створення;

9) дату та час останньої модифікації.
Елементи перерахування 1, 2 та 3 визначають повне ім'я файлу.
При традиційній незв'язної фізичної організації файл може займати кілька рознесених ділянок зовнішньої фізичної пам'яті. У разі розподілу за допомогою списків секторів (блоків) сектори, що належать до одного файлу, містять посилання-покажчики один на одного. Усі вільні сектори диска містяться у списку вільного простору. Подовження або скорочення файлу змінює лише список вільних секторів. Однак логічна вибірка суміжних значень може вимагати тривалих підведення головок дисковода. Зберігання посилань зменшує обсяг пам'яті.
Найбільш загальними операціями роботи з файлами є такі операції:
• пов'язування повного імені файлу із файловими змінними;
• відкриття файлу (наприклад, для запису, лише читання, зміни довжини);
• закриття файлів;
• Встановлення атрибутів файлу.
Закриття файлу є важливою операцією. При її виконанні відбувається фізичне виштовхування інформації із файлового буфера операційної системи на зовнішній носій, а також звільняються ресурси операційної системи.
Операція встановлення атрибутів файлу дозволяє змінювати атрибути файлу, наприклад, встановлювати, що файл може використовуватись тільки для читання тощо.

4.6.2. Логічна організація файлів

Розглянемо можливості логічної організації файлів Turbo Pascal.
Оператори мови Read, ReadLn, Write, WriteLn (при файловій змінній типу Text) забезпечують роботу з файлами єдиного типізованого в мові Pascal виду - текстовими файлами, що являють собою логічно послідовність текстових рядків. Самі текстові файли логічно мають послідовну організацію. Наприклад, щоб прочитати сотий рядок, необхідно до цього прочитати всі попередні рядки. Для текстового файлу в Turbo Pascal є процедура «Append» додавання текстової інформації в кінець текстового файлу. Процедура Append повністю характеризує можливість мінливості текстових файлів (у текстових файлах навіть не можна замінити вміст одного рядка на інший рядок).
Операторами мови Read, Write (файлова змінна має тип File of тип_запису) також можна послідовно записувати у файл або зчитувати з файлу в тій же послідовності один або кілька записів певного типу (фіксованої довжини). Такі файли називають типізованими або файлами у вигляді блокованих записів фіксованої довжини. Якщо записів у типізованих файлах кілька, то за допомогою операції «Seek» можна задати будь-який номер наступного запису, що змінюється або зчитується. Таким чином, реалізовані методи як послідовного, і прямого доступу до інформації файлу, що одночасно утворює комбінований доступ.
*Файлам із довільною організацією*мовою Turbo Pascal відповідають нетипізовані файли, або бінарні. З будь-яким типізованим файлом можна працювати як із нетипізованим файлом.
Нетипізовані файли в Turbo Pascal описуються за допомогою зарезервованого слова «File». Зазвичай роботу з такими файлами здійснюють за допомогою підпрограм BlockRead, BlockWritte, Seek. Також до нетипізованих файлів можуть бути використані всі стандартні засоби роботи з файлами, крім Read, Write, Flush. При використанні процедури Seek кожен блок нетипізованого файлу розглядається як фізичний запис довжиною 128 байт.
Текстові файли Turbo Pascal (як у кодуванні MS DOS, так і в Windows) зазвичай мають розширення (тип) txt і в бінарному (фізичному) поданні є одним записом довільної довжини, що містить послідовність всіх символів рядків, що закінчуються символами «0D16», « 0A16». Останнім символом файлу (необов'язково) може бути символ 1A16, що є ознакою кінця текстового файлу. Символ 0D16 (CR) — повернення каретки без просування паперу. Символ «0A16» (LF) — рух паперу на один рядок донизу.
Таким чином, можна розглядати типізований текстовий файл як нетипізований (бінарний), що складається з одного запису у вигляді масиву символів.
Turbo Pascal практично (за винятком додавання до кінця текстового файлу) не підтримує мінливість структури файлів фізично. Щоб досягти мінливості структури файлів не тільки шляхом повільного копіювання інформації в новий файл з новою структурою, програміст змушений вдаватися до розробки

процедур зміни структури існуючих файлів з використанням низькорівневого програмування на рівні блоків операційної системи. При цьому потрібний високий професіоналізм програміста для забезпечення цілісності файлів, наприклад, при відключенні комп'ютера під час виконання таких процедур.
Уникнути програмування на низькому рівні дозволяє прийом запису змін до тимчасового файлу правок. На логічному рівні старий незмінений файл і короткий файл редагування (або файл додавання в кінець старого файлу) розглядаються як єдиний файл. При виході з програми, а також у певні моменти автоматичного збереження відбувається копіювання з об'єднанням інформації старого файлу та правок файлу в тимчасовий файл. Далі старий файл перейменовується на файл з розширенням імені ВАК. Нарешті, тимчасовий файл перейменовується на робочий файл. Тепер нескладно реалізувати процедури відновлення файлової інформації у разі збоїв апаратури.

4.6.3. Документування файлів

Структура файлів створюється одночасно з виявленням оперативних структур даних та з розробки процедур запису інформації у файл та зчитування інформації з файлу. Опис файлів зазвичай починається із вказівки призначення, повного імені файлу, атрибутів, диспозиції, організації та виду доступу. У документальному описі організації файлів стандартної організації досить згадати тип файлу. Наприклад, файл типу текстовий у кодуванні MS DOS. За потреби можна додатково описати порядок смислових рядків тесту.
Документування порядку проходження інформації у файлах, що складаються зі зблокованих записів фіксованої довжини і з великою кількістю полів, а також документування складних нетипізованих файлів зазвичай виконують трьома способами.
Згідно з першим способом порядок інформації у файлі визначається послідовним розглядом ланцюжків байтів файлу з використанням таблиць.
За другим способом, порядок розміщення інформації у файлі визначається коментованими описами оперативної структури даних мовами програмування, з яких здійснюється запис інформації у файл і які передбачається зчитування інформації з файлу.
Згідно з третім способом опис виконаний за другим способом, доповнюється текстами процедур «читання/запису» файлу.
Практика показала, що використання документації файлів, складеної сторонніми фірмами за другим і особливо за третім способом не викликало труднощів.
"Читання/запис" файлів зі складною довільною організацією, як правило, проводиться послідовними порціями. Перша порція зчитується у статичну запис оперативної пам'яті. Цей запис називають заголовним (header). Вона містить один або кілька байтів ідентифікації, які необхідні для перевірки автентичності файлу (приналежності до конкретних програм). У заголовній інформації може бути вказана версія файлу. Зчитування наступних порцій здійснюється як у статичні, так і динамічні зв'язкові змінні, причому їх довжина може визначатися інформацією, отриманою як з заголовної порції, так і з попередніх порцій.
Розглянемо приклад документування файлу представленої раніше електронної таблиці з допомогою таблиць структури файла. При цьому алгоритми процедур запису інформації до файлу та зчитування інформації з файлу проектувалися одночасно з оперативними структурами електронної таблиці.
При розробці структури файлу було додано такі глобальні описи:

```
Const
{Характеристики файлу}
FILEIDENT = 'My Spreadsheet'; {Ідентифікатор}
FILESEXTENSION = 'MSS'; {Стандартний тип файлу}
Var
FeleName: String; {Ім'я файлу таблиці}
{Видима ширина колонок таблиці}
ColWidth: array [1..MAXCOLS] of Byte;
{Інформація про заповнення таблиці}
LastCol, {Остання заповнена колонка таблиці}
LastRow: Word; {Останній заповнений рядок таблиці}
```

Локальні описи:

```
Var
```

```
EndOfFile; Char; {Ознака кінця текстового файлу}
Col; Word; {Номер колонки клітини}
Row; Word; {Номер стовпця клітини}
Count; Word; {Кількість заповнених клітин таблиці}
Size; Word; {Довжина інформації клітини}
CPtr; TCellPtr; {Покажчик на клітку}
F; File; {Файлова змінна}
Blocks; Word; {Прочитано або записано байт}
```

Файл зберігання електронної таблиці є файлом постійного зберігання, бінарним довільною довжиною; має ім'я, визначене користувачем, але з розширенням імені MSS.

Організація заголовної частини файлу електронної таблиці представлена табл. 4.1.

Таблиця 4.1

Заголовна частина файлу електронної таблиці

Оперативна інформація	Довжина оперативної інформації, байт	Коментар
FILEIDENT	Length (FILEIDENT)	Константа рядка ідентифікації
EndofFile	SizeOf (EndOfFile)	Ознака кінця текстового файлу
LastCol	SizeOf (LastCol)	Остання заповнена колонка таблиці
LastRow	Sizeof (LastRow)	Останній заповнений рядок таблиці
Count	Sizeof (Count)	Число заповнених клітин таблиці на ділянці таблиці (1, 1, LastCol, LastRow)
ColWidth	Sizeof (ColWidth[1] * MAXCOLS)	Вектор ширин колонок таблиці від 1 до MAXCOLS

Запис у файл EndOfFile зі значенням 2610 = 1A16 (Ctrl + Z) забезпечує виведення на екран лише рядка ідентифікації під час перегляду файлу за допомогою більшості програм перегляду текстових файлів.

Під час читання файлу електронної таблиці зчитана інформація першого текстового рядка файлу перевіряється на збіг з FILEIDENT.

Інформація про заповнення таблиці характеризує ділянку таблиці (1, 1, LastCol, LastRow), у якого користувач вніс зміни інформації таблиці.

Значення Count під час запису розраховується з використанням двох вкладених циклів, що задають номери всіх клітин на ділянці таблиці (1, 1, LastCol, LastRow). У циклах значення Count збільшується на одиницю, якщо значення покажчика інформацію клітини ≠ Nil.

У таблиці 4.2 наведено організацію інформації чергової непустої клітини файлу електронної таблиці.

Таблиця 4.2

Інформація чергової непустої клітки файлу електронної таблиці

Оперативна інформація	Довжина оперативної інформації, байт	Коментар
Col	SizeOf (Col)	Номер стовпчика клітини
Row	SizeOf (Row)	Номер рядка клітки
Cells [Col, Row].	Sizeof (Word)	Формат клітини
Size	Sizeof (Size)	Довжина інформації клітини
Фактична інформація клітини	Size	Інформація клітини

Значення Col, Row визначають збережені чи збережені у файлі координати кожної непустої клітини.

Фрагмент коду програми збереження інформації непустої клітини таблиці наведено нижче:

```
if Cells [Col, Row].CellPtr <> nil then
begin
CPtr := Cells [Col, Row].CellPtr;
case CPtr^.Attrib of
TXT : Size := Length (CPtr^.TextStr) + 3;
VALUE : Size := Sizeof (Real) + 2;
FORMULA : Size : = Length (CPtr^.Formula) + Sizeof (Real) + 3;
end; {case}
BlockWrite (F, Col, SizeOf (Col), Blocks);
BlockWrite (F, Row, SizeOf (Row), Blocks);
BlockWrite (F, Cells [Col, Row]. CellFormat, Sizeof (Word), Blocks);
```

```
BlockWrite (F, Size, SizeOf (Size), Blocks);
BlockWrite (F, CPtr^, Size, Blocks);
end;
```

ВИСНОВКИ

• Під структурою даних програми розуміють безліч елементів даних, зв'язків з-поміж них, і навіть характер їх організованості. Структури даних та алгоритми є основою побудови програм.
• Структура даних може бути фізичною та логічною. У загальному випадку між логічною та відповідною їй фізичною структурами є відмінність, внаслідок якої існують правила відображення логічної структури на фізичну структуру.
• Над усіма структурами даних може виконуватися п'ять операцій: створення, знищення, вибір (доступ), оновлення, копіювання.
• Важлива ознака структури даних – характер упорядкованості її елементів. Існує безліч способів упорядкування інформації, серед яких є і загальні, найбільш часто зустрічаються і відомі більшості програмістів.
• Фізичне уявлення може не відповідати логічному уявленню та, крім того, суттєво різнитися у різних програмних системах.
• Багато з розглянутих структур даних можна реалізувати з використанням статичних структур даних, динамічних змінних і динамічних структур даних.
• Файл — впорядкований набір інформації на зовнішньому носії (найчастіше дисковому носії).

Контрольні питання

1. Що таке структура даних програми?
2. Що розуміють під організованістю даних?
3. У якій формі можуть надаватися дані?
4. Що відбиває фізична структура даних?
5. У чому різниця між фізичною та логічною структурами даних?
6. Які операції можуть виконуватись під структурами даних?
7. Наведіть приклади широко відомих структур даних.
8. Чим характеризується статичний масив?
9. Що таке рядок? Які бувають види рядків?
10. Назвіть найпростіший спосіб пошуку елемента.
11. Назвіть основні операції над стеком.
12. Назвіть процедури для роботи із симетричною матрицею.
13. Наведіть приклад реалізації мовою Turbo Pascal структури даних електронної таблиці.
14. Які засоби містить файлова система?
15. Яку інформацію містить дескриптор файлу чи блок керування файлом?
16. З чого зазвичай починається опис файлів?
17. Якими методами зазвичай виконують документування складних нетипізованих файлів?
18. Що таке рефакторинг?
19. У яких випадках може знадобитися рефакторинг імен?