МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

С.В. Коляденко
В.О. Денисюк
Н.П. Юрчук

ДИСКРЕТНИЙ АНАЛІЗ

Частина І

Навчальний посібник

Вінниця – 2020

УДК 519.854 (075.8)
К 62

Рекомендовано Вченою радою
Вінницького національного аграрного університету
Протокол № 6 від 20 грудня 2019 р.

Рецензенти:
О.Н. Романюк – доктор технічних наук, професор, доцент кафедри програмного забезпечення Вінницького національного технічного університету
О.М. Джеджула – доктор педагогічних наук, професор, завідувач кафедри математики, фізики та комп'ютерних технологій Вінницького національного аграрного університету
Л.Б. Ліщинська – доктор економічних наук, професор, завідувач кафедри Вінницького торговельно–економічного інституту КНТЕУ

Матеріали подані в авторській редакції

Коляденко С.В. Дискретний аналіз. Частина І: навчальний посібник. С.В. Коляденко, В.О. Денисюк, Н.П. Юрчук / Вінниця: ВНАУ, 2020. – 161 с.

У посібнику узагальнено теоретичні та практичні матеріали із сортування та пошуку елементів у масивах, зокрема розглянуто основи алгоритмізації, властивості алгоритмів, способи їх представлення, методи сортування масивів, пошук елементів у масивах.

Навчальний посібник призначено для здобувачів вищої освіти галузі знань 05 «Соціальні та поведінкові науки» спеціальності 051 «Економіка» спеціалізації «Економічна кібернетика» першого (бакалаврського) освітнього рівня, але може бути корисним також для тих, хто вивчає курс дискретного аналізу і застосовує його методи у прикладних питаннях, і для тих, хто бажає самостійно оволодіти навичками застосування методів дискретного аналізу.

УДК 519.854 (075.8)
К 62

ЗМІСТ

ПЕРЕДМОВА

Дискретний аналіз подає економіку як первинну сукупність статистичних характеристик з подальшим перетворенням її до неперервних функцій і зв'язків. Ця дисципліна, маючи на меті математичне моделювання економічних процесів, вивчає засоби кількісного аналізу різних дискретних масових явищ, економіку як єдине ціле, зміст і методи обчислення показників соціально-економічних явищ. Отже, предметом дискретного аналізу є дискретні кількісні показники масових соціально-економічних явищ, закономірності їх формування (теорія множин, комбінаторика), взаємозв'язку (математична логіка, теорія графів) і розвитку. Особливістю даної дисципліни є вивчення структур скінченного дискретного характеру, що є характерним для економічних об'єктів. Тому дискретний аналіз нині – важлива ланка економічної освіти. Характерною рисою дискретної математики є відсутність граничного переходу та неперервності, що суттєво відрізняє її від класичної математики.

Метою дисципліни є ознайомлення з основними поняттями, результатами і методами досліджень дискретної математики, навчити ефективно застосовувати зазначений математичний апарат для розв'язання різноманітних практичних завдань.

Дискретний аналіз є відносно молодою наукою, значний інтерес до нього, як науки, пов'язаний з бурхливим розвитком цифрових технологій та впровадженням автоматизованих методів і засобів обробки інформації в усі сфери людської діяльності.

У межах дискретного аналізу виділяють низку тісно пов'язаних між собою розділів, кожний з яких можна вважати самостійним науковим напрямом: математична логіка; проблеми верифікації та оптимізації алгоритмів і програм; теорія автоматів (зокрема й цифрових); бульові функції та оптимізація їх канонічних представлень; теорія чисел і діофантові рівняння; теорія графів, перелічувальні та екстремальні задачі на графах; теорія скінчених груп; теорія складності алгоритмів; ймовірнісна комбінаторика, теорія кодування; комбінаторна топологія та геометричні перелічувальні задачі й оцінки; методи розв'язування задач дискретного і бульового програмування (точні, наближені), двоїсті оцінки для цих задач та ін.

Розвиток елементів дискретного аналізу розпочався ще в античні часи (Піфагор, Діофант, Аристотель, Платон), відродився у 16–17 ст. (П. Ферма, Ґ. Ляйбніц, І. Ньютон, Й. Кеплер та ін.), а деякі розділи сформувалися як самостійні напрями у 18-19 ст. (К. Ґаусс, Дж. Буль, П. Лаґранж, А.-М. Лежандр, Е. Ґалуа, Н. Абель, К. Кляйн, П. Чебишов та ін.). Серед учених, які працювали в Україні в галузі дискретного аналізу, –

П. Чебишов, М. Кравчук, В. Глушков, Л. Калужнін, В. Михалевич, О. Зиков. Так, В. Глушков організував в Інституті кібернетики АН УРСР (Київ) школу теоретичних основ інформатики, у якій дослідження присвячені, зокрема, питанням теорії цифрових автоматів, алгебри перетворення програм, теорії проектування обчислювальних комплексів, алгоритмам автоматизації доведень; виховав плеяду відомих вчених у цій галузі, серед них К. Ющенко, О. Летичевський, Ю. Капітонова, Г. Цейтлін. В. Михалевич спільно зі своїми учнями (Н. Шором, В. Шкурбою, А. Куксою, В. Волковичем) розробив методи послідовної комбінаторної оптимізації для комбінаторних задач оптимізаційного проектування, теорії розкладів. І. Сергієнко створив школу наближення методів розв'язування задач дискретної оптимізації, теорії їх стійкості. І. Коваленко спільно з учнями здійснив аналіз імовірнісних комбінаторних задач, які зустрічаються в теорії кодування, математичній статистиці, теорії надійності. Н. Шор заснував школу недиференційовної оптимізації; його методи застосовані при дослідж. оцінок у комбінаторних задачах і вирішенні проблем теорії складності оптимізац. алгоритмів. В. Трубін запропонував оригін. методи дослідження комбінаторних задач вибору й синтезу складних мереж.

Вітчизняні вчені мають світове визнання досягнень у низці важливих проблем дискретного аналізу, зокрема щодо штучного інтелекту, багатоконвеєрних ЕОМ, складності паралельних алгоритмів, наближених і точних методів бульового, дискретного й мішаного програмування, синтезу мереж, нових моделей і методів оцінок для екстремальних задач на графах, у теорії розкладів, теорії ймовірнісних комбінаторних задач, теорії геометричного проектування.

Прийняття управлінських рішень та отримання оптимальних результатів вимагає від фахівців застосування методів економіко-математичного моделювання, що дозволяють будувати моделі економічних процесів адекватні до реальних. Такий підхід передбачає: поперше, якісний аналіз причинно-наслідкових зв'язків на основі засад теоретичної економіки; по-друге, оцінку характеристик процесу і, наприкінці, створення економіко-математичної моделі з використанням методів та засобів дискретного аналізу [16-19, 30, 31, 34-36].

Вивчення дискретного аналізу здобувачами вищої освіти, забезпечує фундаменталізацію освіти, формує у майбутніх фахівців науковий світогляд і розвиває логічне мислення [9-15].

Загальною метою посібника є поглиблення та закріплення знань з розділу сортування та пошуку елементів у масивах дисципліни «Дискретний аналіз», навчити розробляти алгоритми та програми, за допомогою яких виконується сортування елементів у масивах за певними правилами.

Перша частина посібника містить теоретичні та практичні матеріали із сортування та пошуку елементів у масивах. Перший розділ навчального посібника присвячено викладенню основ алгоритмізації та сутності дефініції «алгоритм» [1-6, 11-15, 20-22, 25]. У другому розділі розглянуто сортування елементів у масивах [1, 11, 22, 25]. Докладно розглядаються методи сортування масивів. Третій розділ містить матеріал із пошуку елементів у масивах [1, 11, 22]. В цьому розділі рзглянуто алгоритми, що реалізують різноманітні методи пошуку заданого елементу серед множини компонент визначеної структури даних.

З метою напрацювання у студентів прикладних вмінь і навичок та закріплення теоретичних знань щодо дискретного аналізу для використання в своїй діяльності методів апарату дискретного аналізу та алгоритмічної мови програмування Turbo Pascal у посібнику приведені завдання, приклади та рекомендації з їх виконання [10-15, 23-29, 32, 33].

Даний посібник сприятиме набуттю студентами необхідних знань, умінь, навичок і компетенцій які допоможуть їм не тільки у вивченні дисциплін загальнонаукового та професійного циклів, а й у вирішенні багатьох практичних завдань.

РОЗДІЛ 1
ОСНОВИ АЛГОРИТМІЗАЦІЇ, СУТІНСТЬ І ВЛАСТИВОСТІ АЛГОРИТМІВ

План (логіка) викладу матеріалу:

1.1. Властивості алгоритмів
1.2. Способи представлення алгоритмів
1.3. Правила оформлення граф-схем алгоритмів
1.4. Базові алгоритмічні конструкції
1.5. Визначення складності алгоритмів
1.6. Програмування мовою Turbo Pascal

Алгоритм – це чітко визначена для конкретного виконавця послідовність дій, які спрямовані на досягнення поставленої мети або розв'язання задачі певного типу [1-6].

У 820 році нашої ери в Бухарі був написаний підручник «Аль-Джабр Ва-аль-Мукабала» («Наука виключення скорочення»), в якому були описані правила виконання чотирьох арифметичних дій над числами в десятковій системі числення. Автором підручника був арабський математик Мухаммед Бен Муса аль-Хорезмі. Від слова «альджебр» у назві підручника пішло слово «алгебра», а від імені аль-Хорезмі – слово «алгоризм», що пізніше перейшло в слово **«алгоритм».**

На сучасному етапі розвитку економіки України зростає роль своєчасного та якісного аналізу діяльності підприємств, яка значною мірою залежить від того, наскільки достовірно вони можуть передбачити перспективи свого розвитку на майбутнє. Ефективним інструментом такого аналізу є економетричні методи та моделі. Економетрія базується на вивченні взаємодії різних економічних процесів і показників та відображення цієї взаємодії у формалізованому виді і побудові моделей. Застосування економетричних моделей в умовах ринкової економіки дає можливість підвищити ефективність використання виробничих потужностей, продуктивність праці та знизити собівартість продукції [36].

1.1. Властивості алгоритмів

1. Зрозумілість. В алгоритмі повинні бути лише операції, які знайомі виконавцеві. При цьому виконавцем алгоритму може бути: людина, комп'ютер, робот тощо.

2. Масовість. За допомогою складеного алгоритму повинен розв'язуватися цілий клас задач.

3. Однозначність. Будь–який алгоритм повинен бути описаний так,

щоб при його виконанні у виконавця не виникало двозначних вказівок. Тобто різні виконавці згідно з алгоритмом повинні діяти однаково та прийти до одного й того ж результату.

4. Правильність. Виконання алгоритму повинно давати правильні результати.

5. Скінченність. Завершення роботи алгоритму має здійснюється в цілому за скінченну кількість кроків.

6. Дискретність. Алгоритм повинен складатися з окремих завершених операцій, які виконуються послідовно.

7. Ефективність. Алгоритм повинен забезпечувати розв'язання задачі за мінімальний час з мінімальними витратами оперативної пам'яті.

1.2. Способи представлення алгоритмів

Алгоритми можуть бути представлені [6, 11]:

1) як система словесних правил (лексикографічний або слівно–покроковий спосіб запису алгоритму),

2) у вигляді таблиці,

3) представлені алгоритмічною мовою у вигляді послідовності операторів (операторний спосіб – програмний код),

4) за допомогою графічного зображення у формі граф–схем (графічний або геометричний спосіб запису алгоритму).

Лексикографічне представлення алгоритмів

Алгоритм записується у вигляді пронумерованих етапів його виконання у довільному викладенні природньою мовою спілкування.

Наприклад. Записати алгоритм знаходження найбільшого загального дільника двох натуральних чисел (алгоритм Евкліда):

1) задати два числа;

2) якщо числа рівні, то взяти будь-яке з них за відповідь і зупинитися, інакше продовжити виконання алгоритму;

3) визначити більше з чисел;

4) замінити більше з чисел різницею більшого і меншого з чисел;

5) повторити алгоритм з кроку 2.

Описаний алгоритм може бути застосованим до будь-яких натуральних чисел і повинен приводити до рішення поставленої задачі.

Недоліки:

- строго не формалізується;
- багатослівність записів;
- можлива неоднозначність тлумачення окремих розпоряджень.

Представлення алгоритмів у вигляді таблиці

Це запис алгоритму у вигляді таблиці (таблична форма запису). Таблиці, що використовуються, можуть бути різними.

Порядок складання табличних алгоритмів:

1) Переписати вираз так, як допустимо у теорії алгоритмів.
2) Визначити порядок дій.
3) Ввести позначення проміжних результатів.
4) Занести одержані дані у таблицю.

Таблиця 1.1

Приклад таблиці алгоритму обчислення R=2a +3b

№ дії	Дія	Величина		Результат
		1	2	
1	*	2	a	K
2	*	3	b	U
3	+	K	u	R

Переваги табличних алгоритмів:

- простота представлення;
- можливість використання без програмування;
- легко супроводжувати (зміни виконуються шляхом коректування бази знань).

Основним недоліком табличних алгоритмів є обмежена сфера використання, оскільки не всі алгоритмічні дії можна виразити за їх допомогою.

Представлення алгоритму алгоритмічною мовою (у вигляді програми)

При представленнях алгоритму у словесній формі, у вигляді граф-схеми або у вигляді таблиці допускається певна свобода дій (гнучкість) при зображенні команд, але у той же час ці представлення є достатньо точними. Причому, настільки точними, що людина (оператор) розуміє суть справи і має можливість виконати алгоритм. Проте на практиці у якості виконавців алгоритмів використовуються спеціальні автомати – комп’ютери. Тому алгоритм, призначений для виконання на комп’ютері, повинен бути записаним на зрозумілій для комп’ютера мові. І тут на перший план висувається необхідність точного запису команд, що не залишає місця для довільного їх тлумачення виконавцем. Отже, мова для запису алгоритмів повинна бути формалізована. Таку мову прийнято називати мовою програмування, а запис алгоритму на цій мові – **програмою для комп’ютера**. Іноді програму називають **програмним**

кодом.

Приклад програми знаходження суми двох чисел c=a+b на мові TurboPascal:

```
program sum;
Var a, b,c: integer;
begin
   writeln (‘ввести значення a, b’);
   c:=a+b;
   writeln (‘сумм чисел дорівнює’, c);
   readln;
end.
```

Переваги програмного коду:

- компактність;
- можливість однозначного запису алгоритму;
- можливість запису з будь–яким ступенем деталізації;
- легко коректувати і доповнювати алгоритм на основі текстових редакторів (легко супроводжувати).

Головний недолік – необхідність знання алгоритмічної мови (мови програмування), яка може бути достатньо складною.

Представлення алгоритму у вигляді граф-схеми

Граф-схема – це спосіб представлення алгоритму у графічній формі у вигляді геометричних фігур, сполучених між собою лініями. Форма блока визначає тип дії, а текст усередині блока дає детальне пояснення конкретної дії (всередині блоків коротко записується зміст дій). Стрілки на лініях, що сполучають блоки схеми, вказують послідовність виконання команд, передбачених алгоритмом.

Граф-схеми, за рахунок наочності спрощують створення ефективних алгоритмів, розуміння роботи вже створених, а як наслідок і їх оптимізацію. Існуючі стандарти на типи блоків дозволяють легко адаптувати алгоритми, створені у вигляді граф-схем до будь-яких існуючих тепер мов програмування.

Зображення блоків у алгоритмі, їх розміри, товщина ліній, кут нахилу ліній тощо, регламентуються Державним стандартом «Схеми алгоритмів, програм, даних і систем», а саме – ГОСТ 19.701–90 (ISO 5807–85). ГОСТ 19.701–90 розповсюджується на умовні позначення (символи) у схемах алгорітмів, програм, даних і систем і встановлює правила виконання схем, які відображують різні види задач обробки даних і

засобів їх рішення. Стандарт не розповсюджується на форму записів і позначень, що містяться усередині символів або поряд з ними та на службові уточнення щодо виконуваних ними функцій. Вимоги стандарту є обов'язковими [7, 8].

1.3. Правила оформлення граф–схем алгоритмів

1) Порядок виконання програми (потік управління) і обробки даних (потік даних) у схемах показуються лініями. Напрямки потоку зліва направо та зверху до низу вважаються стандартними. У випадках, коли необхідно внести велику ясність у схему, на лініях використовуються стрілки. Якщо потік має напрям відмінний від стандартного, стрілки повинні вказувати цей напрям обов'язково.

2) У схемах слід уникати перетину ліній. Лінії, що перетинаються не мають логічного зв'язку між собою, тому зміна напряму потоку у точці перетину не є можливою. Дві та більша кількість вхідних ліній можуть бути поєднані в одну лінію, що виходить. Якщо дві або більше ліній об'єднуються в одну лінію, місце об'єднання повинне бути переміщеним у бік (рис. 1.1).

Рис. 1.1. Об'єднання ліній у граф–схемі алгоритму

3) Лінії у схемах повинні підходити до символу або зліва, або зверху, а виходити, або справа, або знизу.

4) При необхідності лінії у схемах слід розривати для уникнення зайвих перетинів або дуже довгих ліній, а також, якщо схема алгоритму складається, з декількох сторінок. Для цього використовується з'єднувач.

5) Для запису математичних виразів використовуються тільки математичні символи, а не оператори конкретної мови програмування (наприклад, знак рівності «=», а не оператор привласнення « :=»).

6) Блоки у граф-схемі з'єднуються лініями потоків. У кожен блок може входити не менше однієї лінії, з блоку ж (окрім логічного) може виходити лише одна лінія потоку . З логічного блоку завжди виходять дві лінії потоку: одна у випадку виконання умови, інша – при її невиконанні.

На рис. 1.2 представлено основні елементи граф-схем алгоритмів. В таблиці 1.2 представлено зміст позначень блоків на граф-схемах

алгоритмів.

Алгоритм може бути детальним, або спрощеним (деякі зрозумілі блоки можуть не записуватись, інакше алгоритм збільшується у розмірі).

Основні види граф-схем:

- прості (нерозгалужені);
- розгалужені;
- циклічні;
- з підпрограмами;
- змішані.

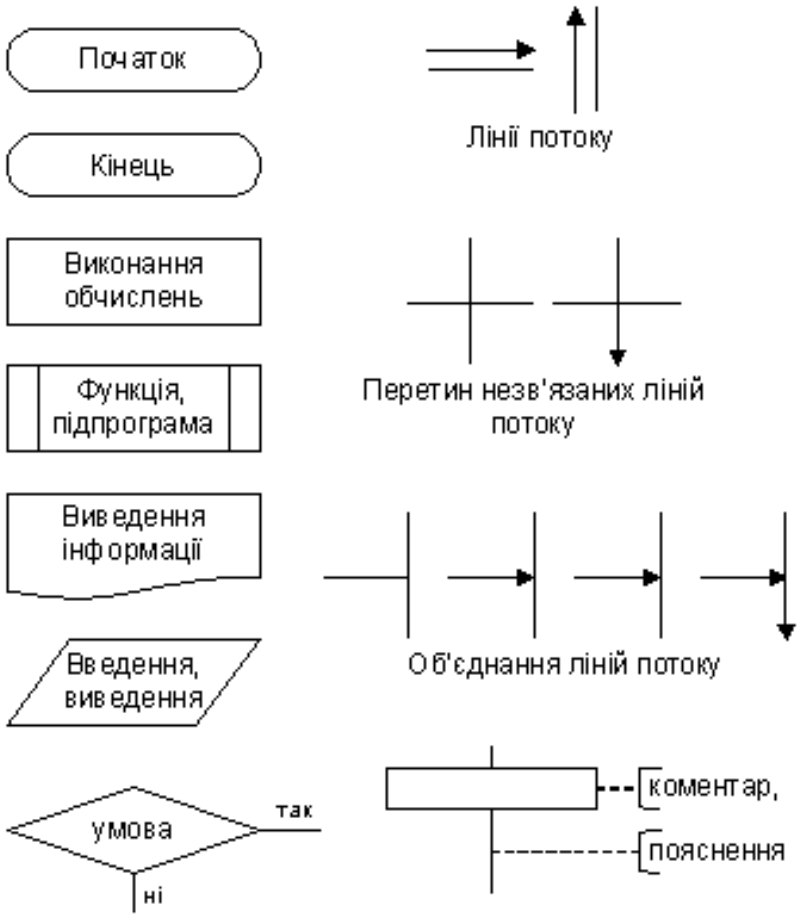
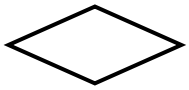


Рис. 1.2. Основні елементи граф-схем алгоритмів

Таблиця 1.2

Зміст позначень блоків на граф-схемах алгоритмів

Позначення	Зміст
	Обчислення або послідовність дій обчислення. Цей символ називається процесом
	Обчислення або послідовність дій обчислення, які визначені в підпрограмі або у модулі. Називається зумовленим блоком

	Перевірка істинності умови, залежно від результату перевірки далі програма виконується по одній із гілок алгоритму. Цей символ називається рішенням (розвилкою).
	Початок циклу, підготовка змінних (наприклад, лічильника циклу для for) для подальших обчислень у циклі. Цей символ називається підготовкою.
	Введення/виведення даних, повідомлень. Цей символ називається символом введення/виведення
	Розрив лінії потоку, використовується при перенесенні схеми алгоритму на іншу сторінку. Цей символ називається з’єднувачем
	Початок/кінець основної програми, вхід/вихід з підпрограми або переривання природнього виконання програми

Слід зауважити, що графічному способу представлення алгоритмів надається перевага через його простоту, наочність та зручність.

1.4. Базові алгоритмічні конструкції

Базові алгоритмічні конструкції – це способи управління процесами обробки даних [1, 6, 11, 15, 25].

Виділяють три базові алгоритмічні конструкції:

1) лінійні алгоритми;

2) алгоритми розгалуженої структури;

3) алгоритми циклічної структури.

Алгоритм називається лінійним, якщо блоки алгоритму виконуються один за одним. Алгоритми лінійної структури не містять умовних і безумовних переходів, циклів.

Лінійні алгоритми наведені на рис. 1.3.

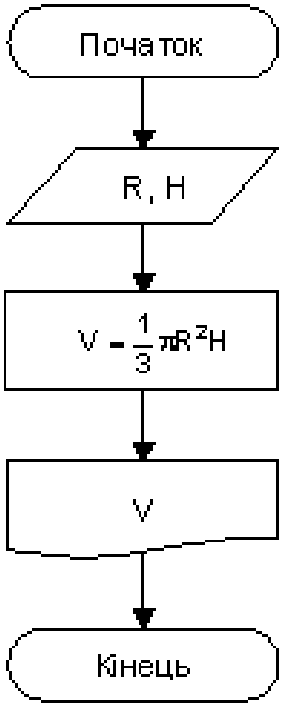


Рис. 1.3. Приклад граф-схеми лінійного алгоритму

Якщо обраний метод розв’язання задачі передбачає виконання різних дій в залежності від значень будь-яких змінних, але при цьому кожна гілка алгоритму в процесі розв’язання задачі виконується не більше одного разу, алгоритм називається розгалуженим.

Алгоритми розгалуженої структури представлені на рис.1.4.

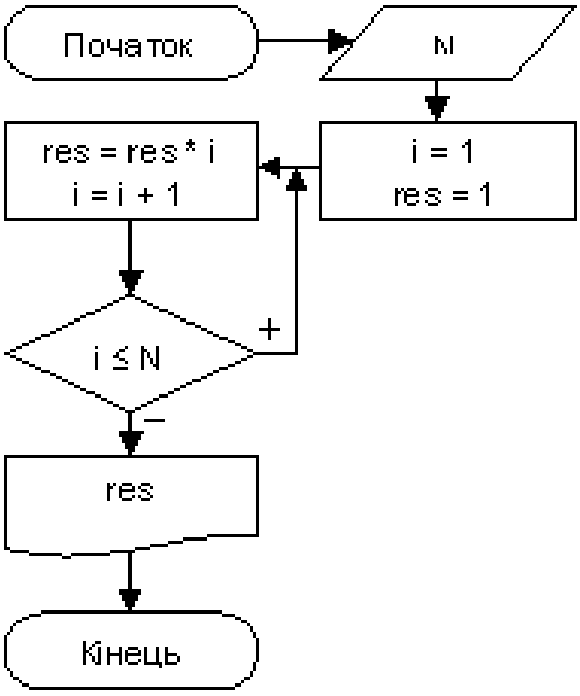


Рис. 1.4. Приклад граф-схеми розгалуженого алгоритму

Алгоритми циклічної структури (рис. 1.5) використовують цикли. Цикл – це команда виконавцеві (компілятору) багаторазово повторити послідовність певних команд. Під час виконання алгоритму циклічної структури відбувається багатократне проходження деяких ділянок алгоритму. Кількість проходжень циклу має бути повністю визначена алгоритмом розв'язання задачі, інакше виникає «зациклювання», при якому процес розв'язання задачі не може завершитися.

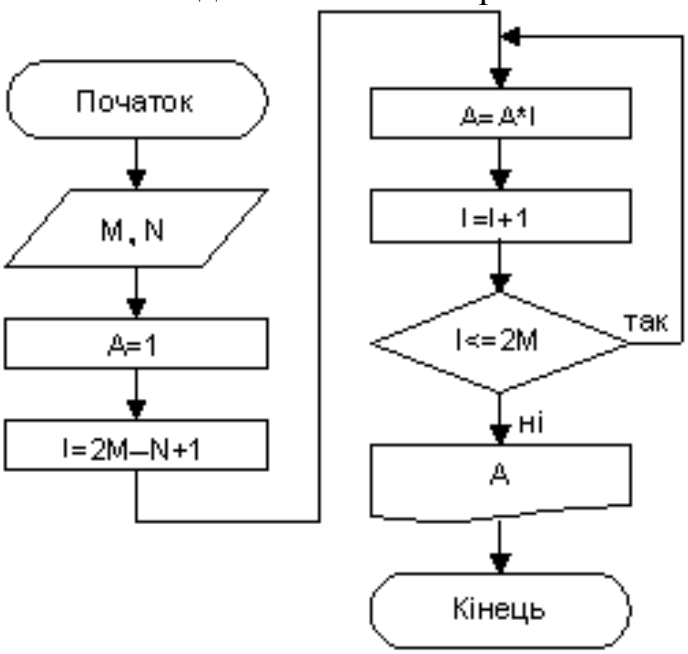


Рис. 1.5. Приклад граф–схеми циклічного алгоритму

Алгоритми розв'язку задач циклічної структури можуть бути такими, що при однократному проходжені циклу деякі ділянки алгоритму виконуються неодноразово, тобто всередині циклу існують інші цикли. Алгоритми такої структури називаються алгоритмами з вкладеними циклами.

1.5. Визначення складності алгоритмів

При розгляді методів використовуються поняття часової ***T*** теоретичної складності алгоритму і просторової ***O*** теоретичної складності алгоритму (надалі опускатимемо слово «теоретична»).

Складність алгоритму – одночлен, що відображує порядок величини необхідного ресурсу (часу/додаткової пам'яті) залежно від розмірності задачі [1].

Наприклад, якщо число тактів (дій), яке необхідне для роботи методу описується як $(11\ n^2+19\ n\ \log(n)+3\ n+4)$, де n–розмірність задачі,

то ми маємо справу з алгоритмом із складністю $T(n^2)$. Фактично, зі всіх доданків залишають тільки ті складові, які дають найбільший внесок при великих n (в цьому випадку рештою доданків можна нехтувати) і ігнорується коефіцієнт перед ним.

У більшості випадків виявляється досить важко знайти точну практичну складність алгоритму (функцію від n, яка дозволяє точно визначити необхідний час роботи/об'єм пам'яті). Проте, теоретичну складність знайти можна і це достатньо просто.

Часова (просторова) складність не дозволяє визначити час (необхідний об'єм додаткової пам'яті) роботи алгоритму, але вона дозволяє оцінити динаміку зростання часу (об'єму додаткової пам'яті), необхідного для роботи методу, при збільшенні розмірності задачі. Наприклад, для алгоритму з часовою складністю $T(n^2)$ при достатньо великих n можна стверджувати, що при збільшенні розміру задачі (при сортуванні – розміру масиву) в 3 рази час роботи алгоритму збільшиться в $3^2=9$ разів.

Якщо операція виконується за фіксоване число кроків, не залежне від розміру задачі, то прийнято писати $T(1)$.

У загальному випадку перед одночленом, що відображує складність, знаходиться деякий коефіцієнт C (const). Тому алгоритми з однаковою складністю можуть сильно відрізнятися часом виконання.

Практично час виконання алгоритму залежить не тільки від об'єму множини даних (розміру задачі), але і від їх значень, наприклад, час роботи деяких алгоритмів сортування значно скорочується, якщо спочатку дані частково впорядковані, тоді як інші методи виявляються нечутливими до цієї властивості. Щоб враховувати цей факт, повністю зберігаючи при цьому можливість аналізувати алгоритми незалежно від даних, розрізняють:

- ***максимальну складність***, або складність у найнесприятливішому випадку, коли метод працює найдовше;
- ***середню складність*** – складність методу в середньому;
- ***мінімальну складність*** – складність в найсприятливішому випадку, коли метод справляється найшвидше.

Алгоритми, що розглядаються, мають часову складність $T(n^2)$, $T(n \log(n))$, $T(n)$. Мінімальна складність якого завгодно алгоритму сортування не може бути меншою ніж $T(n)$. Максимальна складність методу, що оперує порівняннями, не може бути меншою ніж $T(n \log(n))$ [7].

Існують методи, які не порівнюють елементи між собою (один із них – сортування розподілом).

Складність *T(log(n))*

Зі всіх складностей, що розглядаються, ця найменша. Дійсно, при достатньо великому ***n: log(n) < n < n log(n) < n²***.

Ця складність має місце, якщо на кожному кроці методу виконується деяке постійне число дій ***C*** і задача розмірності ***n*** зводиться до аналогічної задачі розмірністю ***n/m***, де ***m*** – ціла позитивна константа:

$$\begin{cases} F(n) = C + F(n/m) \\ F(1) = 0 \end{cases}$$

$$F(n) = C + C + F(n/m^2) = \underbrace{C + C + C + \ldots + C}_{\log_m(n)\_раз} = C * \log_m(n)$$

$\Rightarrow$

Складність *T(n)*

Така складність характерна для звичайних ітераційних процесів, коли на кожному кроці методу задача розмірністю ***n*** зводиться до задачі розмірністю ***n-1***, і при цьому на кожному кроці виконується деяке постійне число дій ***C***, яке не залежить від ***n***:

$$\begin{cases} F(n) = C + F(n-1) \\ F(1) = 0 \end{cases} \Rightarrow \boldsymbol{F(n) = C\,(n-1) = C\,n - C}$$

Прикладом такої складності може слугувати будь–який цикл, де число ітерацій прямо залежить від ***n.*** Деякі методи сортування в кращому разі (якщо масив вже відсортований) виконують порядку ***n*** дій, а сортування розподілом, наприклад, завжди вимагає порядку ***n*** дій.

Складність *T(n*log(n))*

Така складність має місце, якщо на кожному кроці методу виконується деяка кількість дій (***C*n)***, тобто кількість дій прямопропорційно залежить від розміру задачі, і задача розмірністю ***n*** зводиться до задачі розмірністю ***n/m***, де ***m*** – ціла позитивна константа:

$$\begin{cases} F(n) = C * n + F(n/m) \\ F(1) = 0 \end{cases} \Rightarrow \boldsymbol{F(n) = C'\,n\,\log_m(n)}$$

Складність *T(n²)*

Така складність характерна для двох вкладених циклів, число ітерацій яких прямопропорційно залежить від ***n,*** або іншими словами, на

кожному кроці методу задача розмірності ***n*** зводиться до задачі розмірності ***n-1***, і при цьому виконується деяка кількість дій ***(C n),*** тобто прямопропорційно залежне від ***n*** :

$$\begin{cases} F(n) = C * n + F(n-1) \\ F(1) = 0 \end{cases} \Rightarrow F(n)=C'*n^2+C''$$

Алгоритми методів сортування з квадратичною складністю достатньо прості, але малоефективні при великих ***n*** (наприклад, популярне бульбашкове сортування).

При розгляді алгоритмів сортування перш за все нас цікавитиме максимальна і середня складнощі методу, оскільки саме вони, як правило, важливі на практиці. Максимальна складність показує поведінку алгоритму у гіршому разі, а середня складність – найвірогідніша поведінка при збільшенні розміру масиву, що сортується.

1.6. Програмування мовою Turbo Pascal

Мову **Pascal** створив на початку 70–х років професор **Ніколас Вірт** зі Швейцарії [6, 25]. Вона названа на честь французького математика і філософа **Блеза Паскаля** (1623-1662 рр.) – винахідника першої у світі механічної обчислювальної машини. Мова вважається найбільш досконалою, структурованою та придатною для опису алгоритмів порівняно з іншими мовами програмування. Її використовують для розв'язування різноманітних задач. Програми складаються з синтаксичних конструкцій, які називають командами (операторами, вказівками, реченнями).

Середовище програмування Turbo Pascal

Переклад програм, написаних мовою програмування високого рівня (МВР), до яких належить і мова ***Turbo Pascal***, що входить до складу професійного пакету розробки програм ***Borland Pascal with Objects 7.0***, на мову машинних кодів, що виконуються комп'ютером, здійснюється спеціальними програмами, які називаються ***трансляторами.*** За способом роботи транслятори поділяються на ***компілятори*** та ***інтерпретатори.*** Транслятори мови Pascal від найраніших реалізацій до останньої версії ***Turbo Pascal 7.0*** працюють за компілюючим принципом. Спрощена модель компілятора мови ***Turbo Pascal*** представлена на рис.1.6.

програма лексеми внутрішній код машинний код

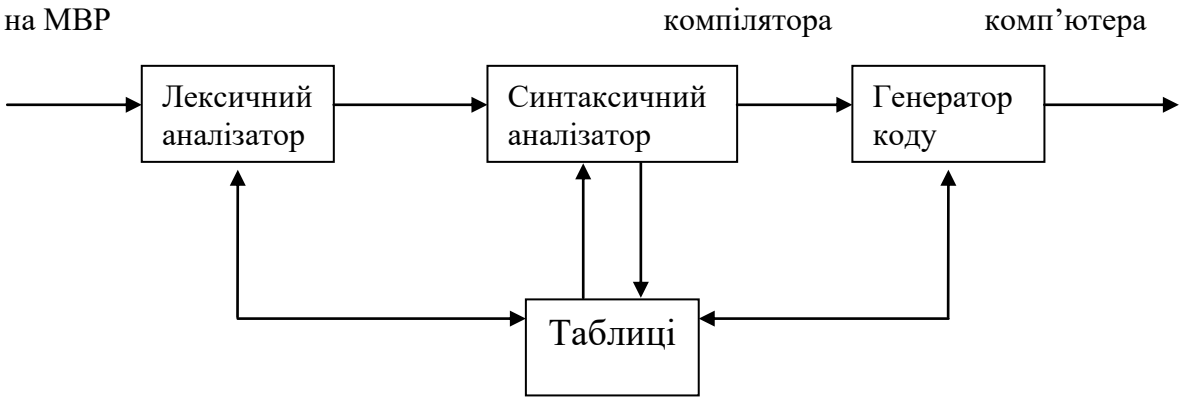


Рис.1.6. Спрощена модель компілятора

Лексичний аналізатор. Початкова програма на *мові високого рівня* (***МВР***) є ланцюжком символів, що утворюється послідовним зчепленням всіх рядків програми. Серед допустимих для мови символів завжди виділяють декілька, так званих, ***символів-роздільників,*** завдяки яким речення початкової програми розбиваються на окремі слова. Такі слова в теорії компіляції називаються ***лексемами.***

Наприклад, речення (оператор) **For i:=1 to n do Writeln(i);**

буде розбито на лексеми:

For, i,:=,1, to, n, do, Writeln,(,i,),;.

Тут *як роздільник використовується символ «пробіл»*. Проте можна відмітити, що між деякими лексемами пропуск не стоїть. Це пов'язано з тим, що *ці лексеми самі є роздільниками* і тому для відділення їх від інших лексем спеціальні символи–роздільники використовувати не обов'язково, хоча й допустимо.

Наприклад**,** теж саме речення без зміни значення можливо записати таким чином:

For i := 1 to n do Writeln (i) ;

Загальне правило. Там, де може стояти один символ–роздільник, допускається будь–яка кількість символів–роздільників. Проте це правило не розповсюджується на лексеми–роздільники, оскільки вони несуть певне смислове навантаження.

Синтаксичний аналізатор на основі синтаксичних правил граматики мови перевіряє коректність запису речень програми і перекладає послідовність лексем у послідовність *внутрішніх кодів компілятора.* Ця послідовність вже відображає порядок дій, які повинен виконати комп'ютер, але ще не є остаточним машинним кодом. У теорії компіляції розроблено декілька різновидів внутрішніх кодів компілятора (тріади, тетради, ПОЛІЗ, дерева, атрибутовані дерева, ***p***–код), проте їх розгляд виходить за рамки спрощеної моделі.

Генератор коду здійснює переклад внутрішнього коду компілятора

в остаточний машинний код комп'ютера.

У процесі роботи всі розглянуті вище блоки компілятора звертаються до загального набору ***таблиць,*** куди поміщується як постійна для трансляції усіх програм інформація (наприклад, таблиця зарезервованих слів), так і інформація, що є індивідуальною для кожної програми (наприклад, таблиці ідентифікаторів, літералів тощо).

Структура професійного середовища розробки програм BORLAND PASCAL WITH OBJECTS 7.0

Попередні продукти фірми ***Borland*** для розробки програм на мові ***Turbo Pascal*** складалися з компілятора і набору модулів, що містять бібліотеки стандартних процедур і функцій. Компілятор мав дві версії, одна з яких працювала в інтегрованому середовищі розробки (***ICP***) (файл ***TURBO.EXE***), а інша – в пакетному режимі (файл ***TPC.EXE***). Новий продукт– компілятор ***Borland Pascal with Objects 7.0 –*** істотно відрізняється від всіх своїх попередників (рис.1.7). Можна відмітити, що взаємозв'язки між блоками даної схеми достатньо складні. Проте насправді, ці взаємозв'язки ще складніше. Наприклад, з ***Program Manager*** оболонки ***Windows*** можна викликати будь-яку з трьох ***ICP***. Зважаючи на перевантаженість схеми (рис. 1.7) різними позначеннями, деякі деталі реалізації були з неї виключені [25].

Компілятор ***Borland Pascal with Objects 7.0*** має три ICP (***TURBO.EXE, BP.EXE, BPW.EXE***) і дві пакетні версії (***TPC.EXE*** та ***BPC.EXE***), які можна згрупувати таким чином:

1) версії компілятора, що працюють під управлінням ***MS-DOS*** в реальному режимі процесора (***TURBO.EXE, TPC.EXE***);
2) версії компілятора, що працюють під управлінням ***MS-DOS*** в захищеному режимі процесора (***BP.EXE*** та ***BPC.EXE***);
3) версія компілятора, що працює під управлінням ***Windows (BPW.EXE).***

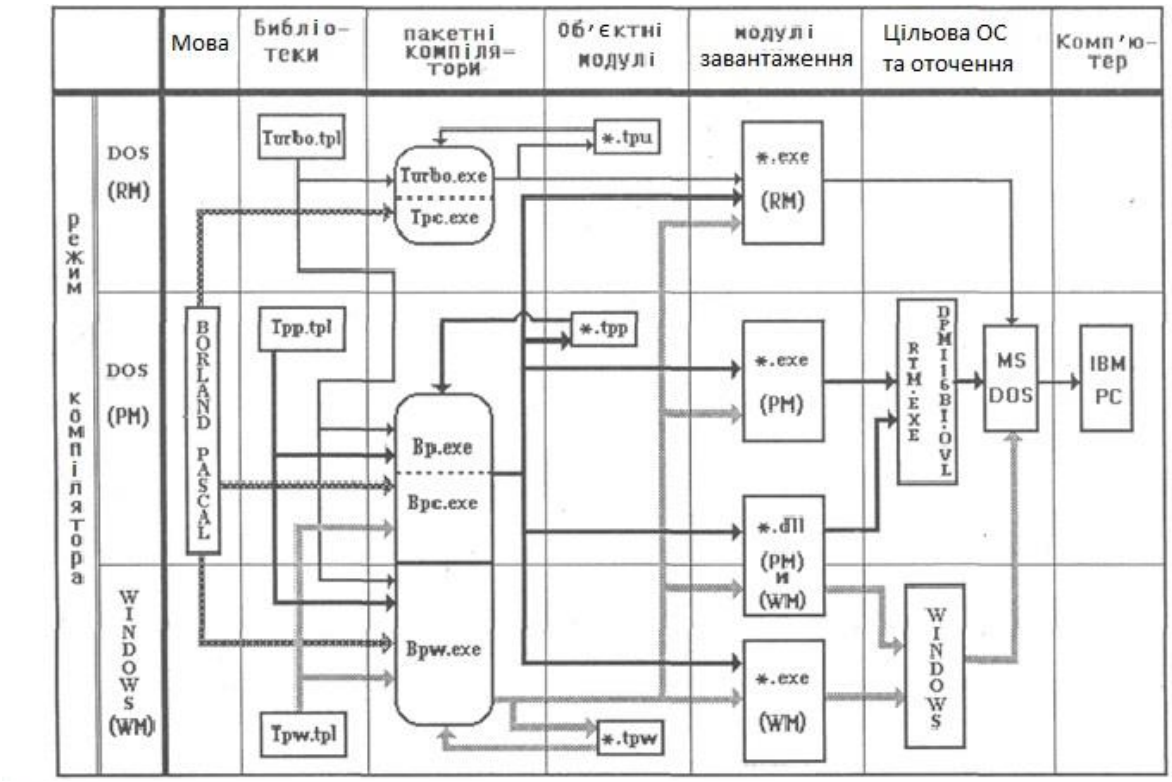


Рис. 1.7. Структурна схема компілятор Borland Pascal with Objects 7.0

Для порівняння можливостей версій компілятора ***Borland Pascal with Objects 7.0*** приведемо деякі їх характеристики по однаковому плану.

Версії компілятора, що працюють під управлінням MS–DOS в реальному режимі процесора (TURBO.EXE, TPC.EXE)

1. Версії компілятора:

а) версія компілятора, що працює в ***ICP (TURBO.EXE);***

б) пакетна версія компілятора ***(TPC.EXE).***

1. Робоче операційне середовище компілятора: ***MS–DOS*** в реальному режимі процесора.

2. Операційні середовища, що підтримуються вихідним кодом компілятора: ***MS-DOS*** реальному режимі процесора.

3. Типи вихідних файлів: ****.EXE, *.TPU.***

4. Файл основних модулів-бібліотек: ***TURBO.TPL***.

Версії компілятора, які працюють під управлінням MS-DOS в захищеному режимі процесора (BP.EXE і BPC.EXE)

1. Версії компілятора:

а) версія компілятора, що працює в ***ICP (BP.EXE);***

б) пакетна версія компілятора ***(BPC.EXE).***

2. Робоче операційне середовище компілятора: ***MS–DOS*** в захищеному режимі процесора.

3. Операційні середовища, що підтримуються вихідним кодом компілятора:

а) ***MS-DOS*** в реальному режимі процесора;

б) ***MS-DOS*** в захищеному режимі процесора;

в) ***Windows***.

4. Типи вихідних файлів: ****.EXE, *.TPU, *.TPP, *.TPW, *.DLL.***

5. Файл основних модулів–бібліотек: ***TPP.TPL***.

Версія компілятора, що працює під управлінням Windows (BPW.EXE)

1. Версії компілятора:

а) версія компілятора, що працює в ***ICP (BPW.EXE);***

б) пакетна версія компілятора (відсутній).

2. Робоче операційне середовище компілятора: ***Windows.***

3. Операційні середовища, що підтримуються вихідним кодом компілятора:

а) ***MS-DOS*** в реальному режимі процесора;

б) ***MS-DOS*** в захищеному режимі процесора;

в) ***Windows.***

4. Типи вихідних файлів: ****.EXE, *.TPU, *.TPP, *.TPW, *.DLL.***

5. Файл основних модулів–бібліотек: ***TPW.TPL***.

Поширені два середовища програмування мовою Паскаль: **Turbo Pascal 7.0** і **Borland Pascal** для **MS–DOS і Windows.** Вони призначені для підготовки текстів програм мовою Паскаль та їхнього виконання. Принципи складання програм (окрім роботи з графікою у **Windows**) для них однакові. Основні файли середовища **Turbo Pascal 7.0** такі: **turbo.exe** (основний виконуваний файл, обсяг 402 Кбайт), **turbo.tpl** (бібліотека, 48 Кбайт, однак може залежати від конфігурації), **turbo.tph** (допомога, 730 Кбайт), **graph.tpu** (модуль для роботи з графікою, 33 Кбайти).

Для входу у середовище треба виконати команду **turbo.exe**. У верхньому рядку екрана буде розташоване головне меню, а в нижньому – опис деяких функціональних клавіш.

Щоб активізувати (увійти в) головне меню, потрібно натиснути на клавішу **F10**. У розпорядженні користувача будуть такі пункти меню:

File – для роботи з файлами;

Edit – для редагування файлу;

Search – для пошуку чи заміни заданого фрагмента тексту;

Run – для виконання програми;
Compile – для компіляції програми та створення exe–файлу;
Debug – для налагодження програми;
Options – для конфігурування середовища;
Window – для конфігурування вікон і роботи з ними;
Help – для надання допомоги.

Технологія трансляції програм

Взагалі електронно–обчислювальна машина (ЕОМ) не розрахована на те, щоб розуміти мову ***LOGO, Pascal*** або інші мови програмування високого рівня. Апаратура розпізнає і виконує тільки машинну мову, програма на якій являє собою не більше ніж послідовність двійкових чисел.

Появу мов програмування було пов'язано з усвідомленням того факту, що переклад алгоритму, написаного на «майже» природній мові, на машинну мову може бути автоматизований і, отже, покладений на плечі машини. Тут важливо розрізняти мову і її реалізацію. Сама мова – це система запису, що регламентується набором правил, що визначають її лексику і синтаксис. Реалізація мови – це програма, яка перетворить цей запис у послідовність машинних команд відповідно до прагматичних і семантичних правил, визначених у мові.

Є два основних способи реалізації мови: ***компілятори*** та ***інтерпретатори***. Компілятори транслюють весь текст програми, написаної на мові програмування, в машинний код в ході одного безперервного процесу. При цьому створюється повна програма в машинних кодах, яку потім можна виконувати без участі компілятора.

Інтерпретатор в кожен момент часу розпізнає і виконує по одному реченню (оператору) програми, по ходу справи перетворюючи речення, що обробляється на машинну програму. Різниця між компілятором і інтерпретатором подібна різниці між синхронним перекладом усної мови і письмовим перекладом тексту.

У загальному випадку будь-яка мова може мати форму як компілятора, так і інтерпретатора, проте в більшості випадків у кожної мови є свій, переважний спосіб реалізації. ***Fortran, Pascal Modula-2*** – в основному компілюють. Такі мови як ***Logo, Fort*** майже завжди інтерпретують. ***BASIC*** і ***Lisp*** широко використовуються в обох формах. Кожний з цих способів має як свої переваги, так і недоліки:

Основні переваги компіляції:

1.Швидкість виконання готової програми

2.Незалежність програми від системи реалізації

Основні недоліки компіляції:

1.Деякі незручності, від яких потерпають програмісти при написанні і налаштуванні великих програм

2.Порівняно великий об'єм пам'яті, який займає компілятор в оперативному запам'ятовуючому пристрої (ОЗП)

Поняття про систему програмування

Розглянемо послідовність змін, що зазнає програма на шляху до виконання (в процесі компіляції).

Текст програми, який написаний на мові програмування, називається початковим модулем. Достатньо складні програми можуть складатися з декількох модулів, що взаємодіють один з одним. Початковий модуль – це вхідний потік для програми ***компілятора***, яка виконує:

1) лексичний аналіз вхідного потоку [блок лексичного аналізу];
2) синтаксичний аналіз вхідного потоку [блок синтаксичного аналізу];
3) генерація машинного коду, який є перекладом початкового модуля на мову ЕОМ в умовних адресах [генератор коду] ;

У результаті цих перетворень на виході отримуємо ***об'єктний модуль***. Навіть якщо ми маємо справу з одним початковим модулем, для успішного виконання програми необхідно «пов'язати» її з деякими іншими програмами (наприклад, із стандартними процедурами введення–виведення, які є в мові програмування). Ці функції виконує програма ***редактор зв'язків***. Вихідний потік цієї програми називають модулем, що завантажується (***модулем завантаження***).

Сучасна технологія застосування ЕОМ вимагає, щоб програму, яка виконується, можна було б розміщувати в довільному місці ОЗП. Тому і завантажувальний модуль написаний в умовних адресах. Розміщенням модуля завантаження у пам'яті займається програма ***завантажувач***.

Зазвичай до складу системи програмування ще включають текстовий редактор та інші сервісні програми. Для управління роботою різних частин системи програмування використовують програму, що управляє, яку називають ***оболонкою***.

Таким чином, для роботи з мовою програмування використовуються спеціальні пакети програм, які називаються інтегрованими системами програмування (ІСП). На рис.1.8 представлено послідовність синтезу програми за допомогою ІСП [11].

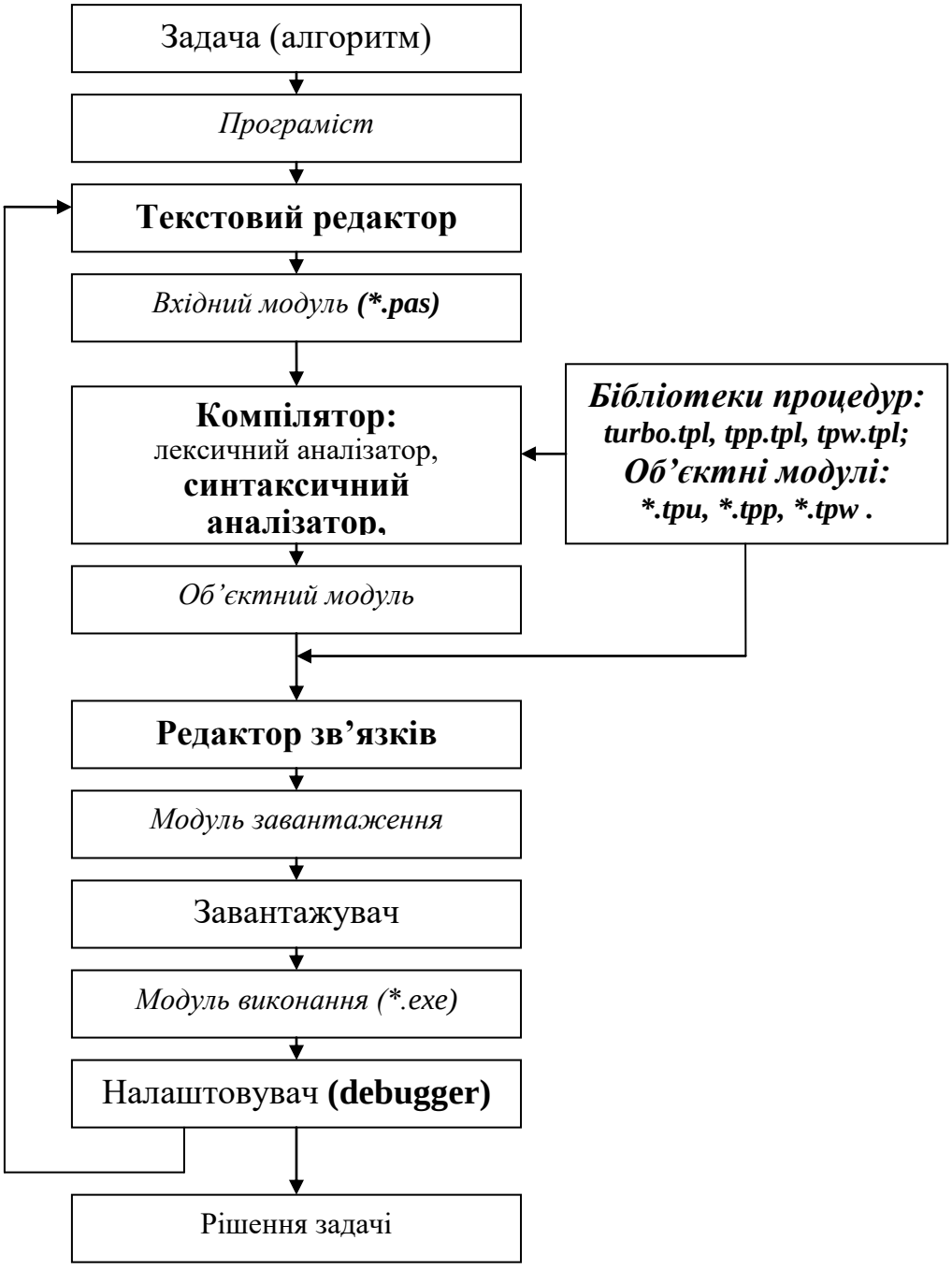


Рис.1.8. Послідовність синтезу програми за допомогою ІСП

Як правило, програми, які тільки що написані, містять безліч помилок.

Помилки бувають:

а) ***синтаксичні*** і ***лексичні*** (виявляються на етапах лексичного і синтаксичного аналізу); наприклад, помилка ***y := sos(x)*** замість ***y := cos(x)*** – лексична, а помилка в операторі ***if x < 0 then y:= 0; else y := 1*** – синтаксична;

б) ***семантичні*** (виявляються на етапі налаштування); наприклад,

помилка в операторі надання значення (привласнення) – ділення на нуль:

x := y; z := 1/(x – y);

в) ***логічні*** (виявляються на етапі контрольних випробувань). До логічних належать такі помилки, в результаті яких програма не робить того, що ми від неї чекаємо; для автоматизації процесу пошуку і усунення семантичних і частково логічних помилок використовуються спеціальні програми, які названі ***налаштовувачами***.

До складу ІСП входять:
1) оболонка,
2) текстовий редактор,
3) компілятор,
4) редактор зв'язків,
5) завантажувач,
6) налаштовувач,
7) бібліотеки стандартних процедур і функцій,
10) сервісні програми.

Елементи мови програмування Turbo Pascal

Алфавіт мови

При записі програм дозволені символи:

– букви латинського алфавіту A-Z (у будь-якому регістрі), а також знак підкреслення _;

– букви кириличного алфавіту А-Я;

– цифри 0-9;

– спеціальні символи > < = + – / * [ ] ( ) { } . , : ; ^ @ ‘ $ # ;

– пари символів (їх не можна розділяти пробілами) < > <= >= := (* *) (..);

– пробіли (розглядаються як обмежники ідентифікаторів, констант, чисел, зарезервованих слів).

Ідентифікатори

Неподільні послідовності символів алфавіту утворюють слова – **ідентифікатори**, використовувані для позначення констант, змінних, процедур, функцій і тощо.

Ідентифікатор може починатися з символу підкреслення, не має містити пробілів і спеціальних символів.

Приклади ідентифікаторів:
name
WorkPhone
_SUM1

Константи

Як константи можуть використовуватися числа, логічні константи, символи і рядки символів.

Цілі числа записуються із знаком чи без знака за звичайними правилами і можуть мати значення від **-2147483648** до **+2147483647**.

Дійсні числа записуються зі знаком чи без нього з використанням десяткової крапки та/чи експонентної частини. Експонентна частина починається символом **e** чи **E**, за яким можуть випливати знаки «+» чи «–» і десятковий порядок. Символ **e (E)** означає десятковий порядок і зміст «помножити на 10 у ступені». Наприклад, запис **3.14E5** означає $\mathbf{3{,}14 \cdot 10^5}$, а запис – 17e-2 – це $\mathbf{-17 \cdot 10^{-2}}$.

Логічна константа – це або слово **FALSE** (неправда) або слово **TRUE** (істина).

Символьна константа – це будь–який символ, укладений в апострофи:

‘z’ – символ **z**;

‘ф’ – символ **ф**.

Рядкова константа – послідовність символів, укладена в апострофи. Якщо у рядку потрібно вказати сам символ апострофа, він подвоюється, наприклад:

‘ Це – рядок символів ‘;

‘ That’’s string’.

Рядок символів може бути порожній, тобто не мати ніяких символів у її апострофах, що обрамляють.

Вирази

Вираз задає порядок виконання дій над елементами даних і складається з операндів (констант, змінних, функцій, круглих дужок і знаків операцій).

Дії у виразі виконуються зліва направо у порядку убування:

1) **Not** (логічне заперечення);

2) ***** (множення), **/** (ділення), **div** (цілочисельне ділення), **mod** (цілочисельне ділення із залишком за модулем), **and** (логічна операція **«Так»**);

3) + (додавання), – (віднімання), **or** (логічна операція **«Або»**);

7) операції відносин: = (дорівнює), <> (не дорівнює), < (менше), > (більше),

< = (менше чи дорівнює) , > = (більше чи дорівнює).

Для зміни порядку виконання дій використовуються круглі дужки. Число дужок, що відкриваються, дорівнює числу що закриваються. Будь-який вираз в дужках обчислюється раніше, ніж виконується операція, що передує дужкам. Усі частини виразу записуються в один рядок, наприклад:

(a+b*x)/(c+d)

У вирази можуть входити функції. Функції, які найчастіше використовуються називають стандартними. Для роботи з ними не треба ні замовляти бібліотеку, ні описувати їх попередньо у програмі. Приклади стандартних математичних функцій:

ABS(x) – модуль x (| x |);
SQR(x) – квадрат числа x (x^2);
SQRT(x) – квадратний корінь з x;
LN(x) – натуральний логарифм від x (ln x);
EXP(x) – e у ступені x (e^x);
SIN(x) – синус x (sin x);
COS(x) – косинус x (cos x);
ARCTAN(x) – арктангенс x (arctg x).

Аргумент цих функцій може бути як дійсним, так і цілим. Результат – завжди дійсний.

Типи даних мови програмування Turbo Pascal

Будь–які дані, тобто константи, змінні, значення чи вирази функцій в Turbo Pascal характеризуються своїми типами. Тип визначає безліч припустимих значень, що може мати той чи інший об’єкт, а також безліч припустимих операцій, що застосовні до нього. Усі типи даних розділяються на дві групи – прості і складені.

До простого (скалярного) типу відносяться, наприклад:

INTEGER – дані цього типу можуть приймати тільки цілі значення (позитивні, негативні, 0) у діапазоні від **–32768** до **+32767**;

REAL – величини цього типу можуть приймати тільки речовинні значення (числа з дробовою частиною, ціла частина від дробової відокремлюється крапкою);

BOOLEAN – логічний тип, приймає два значення TRUE (істина) і FALSE (неправда);

CHAR – символ.

Прикладом складеного (структурованого типу) може бути тип **STRING** – рядок символів. Використовується для обробки текстів.

Структура програми мовою програмування Turbo Pascal

Узагальнена (розширена) структура програми мовою програмування **Turbo Pascal** представлена на рис. 1.9.

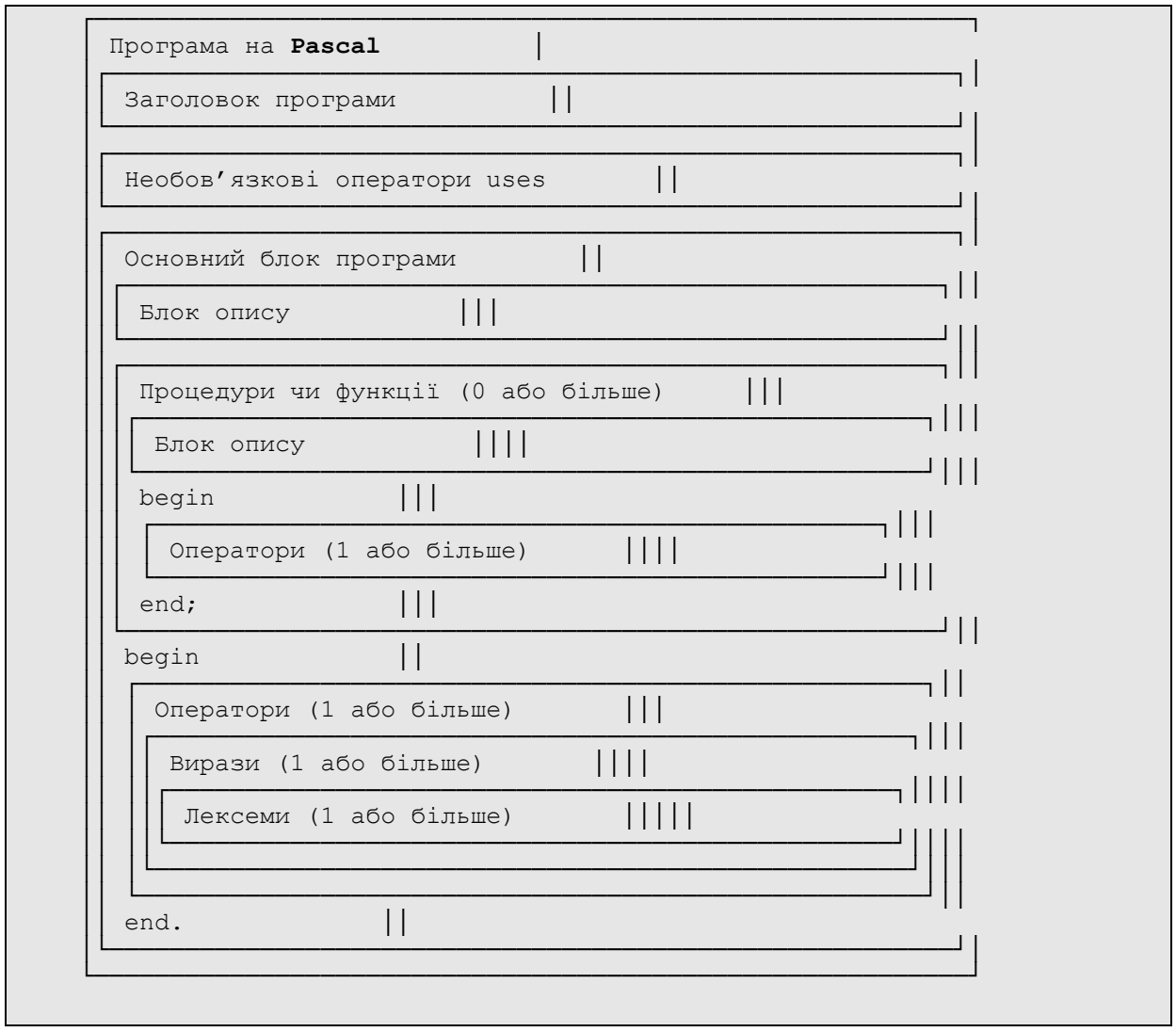


Рис. 1.9. Розширена діаграма програми на **Turbo Pascal**.

Програму складають заголовок програми, необов'язковий оператор **uses** (його розглянемо нижче) і основний блок програми. У основному блоці можуть бути присутніми більш дрібні блоки процедур і функцій. Хоча на діаграмі це не показано, процедури чи функції можуть бути вкладеними в інших процедури або функції. Іншими словами, блоки можуть містити інші блоки.

У поєднанні з іншими лексемами та пробілами лексеми можуть утворювати вирази, що формують оператор. Оператори, у свою чергу, в поєднанні з розділом описів утворюють блоки основної програми або блок в процедурі або функції.

У своїй найпростішій формі програма Borland Pascal складається з заголовка програми, що іменує програму, і основного програмного блоку, що виконує програму:

```
<Заголовок програми>
{Блок описів}
BEGIN
        {Розділ операторів, що виконуються,}
END.
```

В основному програмному блоці знаходиться секція коду між ключовими словами **begin** та **end.** Приведемо просту програму, яка це ілюструє:

```
program Privet;
      begin
        Writeln('Ласкаво просимо в Turbo Pascal');
end.
```

Перший рядок – це заголовок програми, що іменує дану програму. Інша частина програми – це вихідний код, який починається ключовим словом **begin** та закінчується словом **end**. Хоча дана конкретна програма містить тільки один рядок, рядків може бути багато. У будь-якій програмі **Borland Pascal** усі дії виконуються між **begin** та **end.**

У **заголовку програми** вказується ім'я програми. Загальний вид заголовка: **program n;** тут **n** – ім'я програми.

Заголовок програми необов'язковий, його можна опускати без наслідків для програми.

У **блоці описів** вміщуються ідентифікатори типів, констант, змінних, а також мітки, процедури і функції. Блок описів може складатися з п'яти розділів, що мають випливати в чітко визначеному порядку:

1) розділ міток (**label**);
2) розділ констант (**const**);
3) розділ типів (**type**);
4) розділ змінних (**var**);

5) розділ процедур і функцій.

У **розділі міток** вміщується опис міток **(label)** програми. Оператор, що виконується, може бути позначений міткою – позитивною константою, що містить не більш 4-х цифр. Мітка відокремлюється від оператора двокрапкою. Усі мітки, що зустрічаються в програмі, мають бути описані у розділі **label**. Загальний вигляд:

label l1, l2, l3…;

тут **l1, l2, l3…**-мітки.

Приклад: **label 20;**

Нехай оператор **a:=b;** має мітку **20**. Тоді цей оператор виглядає так:

20: a:=b ;

У **розділі констант** вміщується опис констант програми (**const**). Якщо в програмі використовуються константи, що мають досить громіздкий запис (наприклад, число π з 8–ма знаками), або змінні константи (наприклад, для завдання варіанта програми), то такі константи звичайно позначаються якими–небудь іменами й описуються в розділі const. Це робить програму більш наочною і зручною при налагодженні і внесенні змін.

Загальний вид:

const a1 = з1; a2 = з2; …

Тут **a1, a2, …** – ім’я констант, **з1, з2,** ... – значення констант.

Приклад: **const pi=3.14; c=2.7531;**

У **розділі типів** вміщується опис типів змінних програми (**type**). Якщо в програмі вводиться тип, відмінний від стандартного, то цей тип описується в розділі **type**:

type t1=<вид типу>;

t2=<вид типу>;

. . . . . .

де **t1** та **t2** – ідентифікатори типів, що вводяться.

Приклад. **Type color=(red, yellow, green, blue);**

Тут описаний тип **color**, що задається перерахуванням значень.

У **розділі змінних** вміщується опис змінних програми **(var).** Вводиться ім’я кожної змінної і вказується, до якого типу ця змінна належить:

var v11, v12, …: type1;

v21, v22, …: type2; …

Тут **v11, v12, …** – імена змінних; **type1** – тип змінних **v11, v12, …;**

type2 – тип змінних **v21, v22, ...**

Приклад: **var k,i,j:integer; a,b:real;**

У **розділі процедур і функцій** вміщується опис процедур і функцій, які використовуються у програмі. Ті алгоритми, що оформляються як підпрограми (процедури і функції) містяться в головній програмі після розділу **var** і перед **begin** програми. Код між останніми операторами **begin** та **end** програми керує логікою програми. У дуже простій програмі у цій секції коду може міститися усе, що вам потрібно. У більш великих і складних програмах розміщення в цій секції всього програмного коду може погіршити читання і розуміння програми. До того ж її буде складніше розробляти.

Процедури і функції дозволяють розділити логіку програми на більш дрібні і керовані фрагменти, що аналогічно підпрограмам у інших мовах програмування.

У **розділі операторів** вміщується опис операторів, які використовуються у програмі (складають зміст програми). Розділ операторів програми починається з ключового слова **begin** і закінчується словом **end**, після якого повинна стояти крапка (**end**.). Розділ операторів – це частина програми, що виконується. Вона складається з операторів.

Коментар – це довільна послідовність будь–яких символів, що пояснює текст програми. Коментар дозволяється вставляти в будь–яке місце програми, де за змістом може стояти пробіл. Як обмежники коментарю використовуються фігурні дужки « { « і» } «, а також пари символів: « **(*** « – ліворуч від коментаря і « ***)** « – праворуч від нього:

{ Це коментар }
(* Це теж коментар *)

Оператор присвоювання

Під операторами у мові програмування **Turbo Pascal** мають на увазі опис дій. Оператори відокремлюються один від одного крапкою з комою. Якщо оператор знаходиться перед **end, until** чи **else**, то в цьому випадку крапу з комою не ставлять.

Загальний вид оператора присвоювання:

v:=a;

тут **v** – змінна, **a** – вирази, **:=** – операція присвоювання, читається так – *«надати змінній* ***v*** *значення* ***a****»* Вирази **a** може містити константи, змінні, назви функцій, знаки операцій і дужки. В операторі **v:=a** змінна **v** і вирази

a повинні мати однаковий тип.

Приклади:

f:=3*c+2*sin(x); або **x:=x+1;**

Зауваження. Дозволяється привласнювати змінній типу **real** значення виразів типу **integer**. Проте не можна привласнювати змінній типу **integer** вирази типу **real**.

Оператори

Програмний код між **begin** і **end** містить оператори, які описують дії програми. Ця частина називається операторною частиною програми. Приведемо приклади операторів:

```
A := B + C; { присвоїти значення }
Calculate(Length, Height); { активізувати процедуру }
if X < 2 then Answer := X * Y; { умовний оператор }

begin       { складений оператор }
X := 3;
Y := 4;
Z := 5;
end;

while not EOF(InFile) do  { оператор циклу }
begin
ReadLn(InFile, Line);
Process(Line);
end;
```

У простих операторах можна надавати значення, активізувати процедуру чи функцію або передавати управління на іншу частину коду. Структурні оператори можуть бути складеними і містити декілька операторів, оператор циклу або умовний оператор, який керує логікою програми, а також оператори **with**, що спрощують доступ до даних у записах.

Вирази

Оператор **Turbo Pascal** складається із виразів. Вирази можуть складатися з операндів та операцій. Звичайно у виразах виконуються порівняння, арифметичні або логічні операції.

Вирази **Turbo Pascal** складаються з простіших виразів чи комбінації операндів і операцій. Вирази можуть бути достатньо складними. Наведемо деякі приклади виразів:

```
X + Y
Done <> Error
I <= Length
–X
```

Лексеми

Лексеми – це найменші значимі елементи у програмі **Turbo Pascal**. Оператори складаються із виразів, які в свою чергу складаються із лексем (рис.1.10) (лексеми утворюють операнди і вирази). Лексеми – це спеціальні символи, зарезервовані слова, ідентифікатори, мітки і рядкові константи.

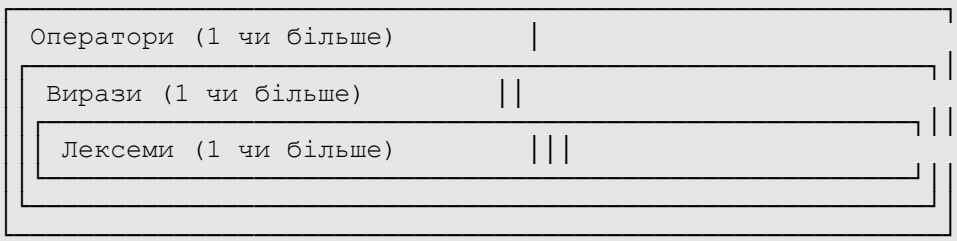


Рис. 1.10. Діаграма оператора.

Приклади лексем Turbo Pascal:

```
function     { зарезервоване слово }
  (      { спеціальний символ }
  :=     { спеціальний символ }
Calculate    { ідентифікатор процедури }
        9                    { число }
```

Типи, змінні, константи та типізовані константи

Змінна містить значення, яке може буте змінено. Кожна змінна має мати тип. Тип змінної визначає безліч значень, які може мати змінна. Наприклад, у наведеній нижче програмі описуються змінні **X** та **Y**, що мають тип **Integer**. Таким чином, **X** та **Y** можуть містити тільки цілі значення (числа). Якщо у програмі буде спроба привласнити цим змінним

значення іншого типу, то **Pascal** повідомить про помилку.

```
program Example;
const   A = 12; {константа A дорівнює 12 та не змінює своє значення}
  B: Integer = 23; {типізована константа B отримує початкове значення}
Var  X, Y: Integer; { змінні X та Y мають тип Integer }
    J: Real;  { змінна J має тип Real }
begin
   X := 7;   { змінній X надається значення }
   Y := 7;   { змінній Y надається значення }
   X := Y + Y; { значення змінної X змінюється }
   J := 0.075;  { змінній J надається значення типу Real }
end.
```

У цій простій програмі змінній **X** спочатку надається значення **7**; двома операторами нижче їй надається нове значення: **Y + Y**. Як можна бачити, значення змінної може змінюватися.

A – це константа. Програма призначає їй значення **12**, і це значення змінюватися не може – у процесі виконання програми воно залишається постійним.

B є типізованою константою. Їй надається значення при описі, але надається також тип **Integer**. Типізовану константу можна розглядати як змінну з початковим значенням. Пізніше програма може змінити первинне значення **B** на якесь інше значення.

Процедури і функції можуть містити розділи описів змінних також як програми і модулі.

Модулі

Програма на **Turbo Pascal** може використовувати блоки програмного коду у програмних модулях (рис.1.6). Модуль (**unit**) можна розглядати як міні-програму, яку може використовувати прикладна програма. Як і програма, він має заголовок (який називається заголовком модуля) і основний блок, обмежений **begin** та **end**.

Основний блок будь-якої програми **Turbo Pascal** може включати у себе рядок, що дозволяє програмі використовувати один або більше модулів. Наприклад, якщо програма з іменем **Colors** хоче змінювати кольори тексту, що виводиться на екран, то програма може використовувати стандартний модуль **Crt**, що є частиною бібліотеки

системи **Turbo Pascal**:

```
rogram Colors;
  uses Crt;
begin
   …
end.
```

Рядок **uses Crt** повідомляє **Pascal**, що потрібно включити модуль **Crt** у програму, що виконується. Крім усього іншого, модуль **Crt** містить увесь необхідний код для зміни кольору у програмі. Шляхом простого включення **uses Crt** програма може використовувати весь код, що міститься в модулі **Crt.** Тому оператор **uses** називають також оператором використання. Якби помістити увесь код, необхідний для реалізації функціональних можливостей **Crt**, у програму, то це зажадало б великих зусиль і відвернуло би увагу від основної мети програми.

Бібліотека системи програмування **Pascal** містить декілька модулів, які є вельми корисними. Наприклад, завдяки використанню модулів **Dos** або **WinDos**, програма може отримати доступ до декількох підпрограм операційних системи і підпрограмам роботи з файлами.

Також системи програмування **Pascal** надає можливість писати свої власні модулі. Їх краще за все застосовувати для розділення великих програм на логічно пов’язані фрагменти. Програмним кодом, який уміщений у модуль, може користуватися будь–яка програма. Потрібно написати початковий код тільки один раз, а потім можливо багато разів його використовувати.

Процедура введення інформації

Загальний вигляд:

Read (v1, v2, …,vn); чи **Readln (v1, v2, …,vn);**
тут **v1, v2, …,vn** – ідентифікатори змінних.

Значення змінних вводяться з клавіатури і мають відповідати типам змінних. У випадку використання процедури **readln**, після введення відбувається перехід на наступний рядок.

Процедура виведення інформації на друк

Загальний вигляд оператора:

write(p_1, p_2, …, p_n); чи **writeln(p_1, p_2, …, p_n);**

Тут **p_1, p_2, …, p_n** – список виразів, значення яких виводяться на друк.

Оператор **write** залишає курсор наприкінці виведеного рядка тексту. При використанні процедури **writeln**, після друку відбувається перехід на наступний рядок. Крім значень виразів, на друк можна виводити і довільний набір символів, уміщений в апострофи, наприклад: **writeln('p=',p);**

Цей оператор виконується так: спочатку виводяться символи, укладені в апострофи. Потім виводиться значення змінної **p**, наприклад, **13.5**. На екрані в результаті роботи оператора з'явиться: **p=13.5**

Приклад 1.

Обчислити довжину кола радіуса 5,785.

```
program t10;   (* Програма обчислення довжини кола*)
  const r=5.785;
   var l:real;
begin
        l:=2*3.1416*r;
        writeln(' l=',l);
end.
```

або

```
program t11;
  var r,l:real;
begin
        readln(r);
        writeln(' l=',2*3.1416*r);
end.
```

Існує можливість задати ширину поля (число позицій) **M** для виведеної величини **P**: **Write (P1:M1, P2:M2, …PN:MN);**

Для дійсних чисел можна задавати поле, як **M** і **N**, де **M** – загальне число позицій, що відводяться під усе число, **N** – число позицій під його дробову частину.

Наприклад: **Write (P:10:2);**

Тут під **P** приділяється **10** позицій, **2** з них під дробову частину.

Приклад 2.

Ввести з клавіатури два цілих числа, знайти результат ділення першого числа на друге та вивести числа і отриманий результат на екран у вигляді таблиці.

```
Program Input_Output; {Вводить два цілих числа, виводить}
        { результат ділення 1–го на 2–ге   }
Var
  n1,n2:Integer;    {n1 і n2 – числа, що вводимо – цілі}
   x:real;      {x – результат}
Begin
        Write('n1='); {Повідомлення про введення n1}
        ReadLn(n1);   {Введення n1}
        Write('n2='); {Повідомлення про введення n2}
        ReadLn(n2);   {Введення n2}
        X:=n1/n2;     {Обчислення результату}
        WriteLn('————————————'); {Друк таблиці}
        WriteLn('| n1 | n2 | Частка |');
        WriteLn('————————————');
        WriteLn(n1:8,n2:8,x:8:4);     {Висновок n1, n2 і x}
        WriteLn('————————————');
End.
```

Методичні рекомендації для комп'ютерного практикуму

1. Запуск програми **Turbo Pascal** виконується кожним зі стандартних способів запуску, передбачених у ОС Windows (наприклад, за допомогою ярлика на робочому столі).

Для переходу до вибору команд головного меню використовується клавіша **F10**. Для повернення в режим редагування потрібно натиснути клавішу **ESC**.

Для одержання довідки використовуються клавіші:

F1 – одержання контекстно–залежної довідки;

SHIFT+F1 – вибір довідки зі списку доступних довідкових повідомлень;

CTRL+F1 – одержання довідки про потрібну стандартну процедуру, функцію, про стандартну чи константу змінної.

2. При рішенні задач можна скористатися прикладами програм, що наведені вище. Текст програми набирається у текстовому редакторі

середовища **Turbo Pascal.**

Після заповнення чергового рядка варто натиснути клавішу **ENTER**, щоб перевести курсор на наступну рядок.

Команди редактора Turbo Pascal, що використовуються найчастіше

Переміщення курсору

Page Up – на сторінку вверх;
Page Down – на сторінку вниз;
Home – у початок поточної рядка;
End – у кінець поточного рядка;
Ctrl+ Page Up – у початок тексту;
Ctrl+ Page Down – у кінець тексту.

Команди редагування

Backspase – стирає символ ліворуч від курсору;
Delete – стирає символ, на який указує курсор;
Ctrl+Y – стирає рядок, у якій розташований курсор;
Enter – уставляє новий рядок, розрізає старий;
Ctrl+Q L – відновлює змінений рядок (діє, якщо курсор не залишав рядок після його зміни).

Робота з блоками

Ctrl+K B – починає виділення блоку;
Ctrl+K K – закінчує виділення блоку (крім того, блок можна виділити за допомогою миші);
Ctrl+K Y – знищує виділений блок;
Ctrl+K З – копіює блок;
Ctrl+K V – переміщає блок на нове місце;
Ctrl+K W – записує блок у файл;
Ctrl+K R – читає блок з файлу;
Ctrl+K P – друкує блок.

Набраний текст програми запишіть у файл. Клавішею **F2** викликається вікно діалогу, у якому варто задати ім’я файлу.

Після підготовки тексту програми потрібно спробувати виконати її, тобто відкомпілювати програму, зв’язати її (якщо це необхідно) з бібліотекою стандартних програм і функцій, завантажити в оперативну пам’ять і передати їй керування. Ця послідовність дій – прогін програми – здійснюється після натискання клавіш **CTRL+F9**.

Якщо в програмі немає синтаксичних помилок, то всі дії

виконуються послідовно одна за одною, при цьому в невеликому вікні повідомляється про кількість відкомпільованих рядків і обсяг доступної пам'яті. Перед передачею керування завантаженій програмі середовище виводить на екран вікно прогону програми, а після завершення роботи відновлює на екрані вікно редактора.

Якщо на якомусь етапі середовище програмування виявить помилку, то воно припиняє подальші дії, відновлює вікно редактора і поміщає курсор на той рядок програми, при компіляції чи виконанні якої виявлена помилка. При цьому у верхньому рядку редактора з'являється діагностичне повідомлення про причину помилки. Усе це дозволяє налагодити програму, тобто усунути в ній синтаксичні помилки і переконатися в правильності її роботи. Деякі повідомлення про помилки, їхній переклад і пояснення наведені у додатку В.

За оператором **read** (чи **readln**) викликається вбудована процедура введення даних і програма зупиняється в очікуванні введення. Необхідно набрати на клавіатурі потрібні дані і натиснути клавішу **ENTER**.

За допомогою клавіш **ALT+F5** у будь–який момент можна переглянути дані, видані на екран у результаті прогону програми.

Контрольні запитання і завдання для самоперевірки

1. Опишіть властивості алгоритмів.
2. Охарактеризуйте способи представлення алгоритмів.
3. Які існують правила оформлення граф–схем алгоритмів?
4. Охарактеризуйте алгоритмічні конструкції.
5. Опишіть, як визначити складність алгоритмів.
6. Що таке ідентифікатори та яке їх призначення?
7. Які типи змінних використовуються в мові Turbo Pascal?
8. Що таке масив даних?
9. Яка структура даних утворює масив?
10. Які можуть існувати типи масивів?
11. Якого типу мають бути індекси масиву?
12. Охарактеризуйте алгоритмічні мови програмування.
13. Назвіть призначення та способи подання алгоритмів.
14. Перелічіть блоки, що використовуються в схемах алгоритму.
15. Назвіть базові алгоритмічні структури.

Тестові завдання для самоконтролю

1. Алгоритм – це:

a) зрозуміле і точне розпорядження на виконання послідовності дій, направлених на досягнення визначеної мети;

b) точне розпорядження на виконання дій, направлених на досягнення визначеної мети;

c) зрозуміле розпорядження на виконання дій для досягнення визначеної мети;

d) матеріальний або віртуальний обє'кт, який використовується замість об'єкта–оригіналу чи явища (процесу) при його дослідженні.

2. На етапі постановки задачі потрібно:

a) на основі побудованої математичної моделі розробити алгоритм;

b) з'ясувати, що дано і що треба знайти, розробити алгоритм;

c) на основі побудованої математичної моделі розробити програму;

d) розбивка алгоритму на окремі елементарні дії (команди), що легко виконуються виконавцем.

3. Властивість алгоритму, який повинен бути однозначно витлумачений називається:

a) результативність;

b) дискретність;

c) визначеність;

d) рекресивність/

4. Яке позначення має структура алгоритму розвилка?

a) ромбом;

b) квадратом;

c) овалом;

d) паралелограмом.

5. До форм подання алгоритму належать:

a) словесний, схеми, блок–схеми, алгоритмічна мова, мова програмування;

b) словесний, усний, схеми, алгоритмічна мова, мова програмування;

c) малюнки, схеми, блок–схеми, алгоритмічна мова, мова програмування;

d) словесний, усний, схеми.

6. Якщо складений алгоритм забезпечує розв'язання не однієї окремої задачі, а розв'язує широкий клас задач даного типу, то ця властивість алгоритму:

a) детермінованість;
b) скінченість;
c) масовість;
d) інваріантність.

7. На етапі розробки алгоритму потрібно:
a) на основі побудованої математичної моделі розробити алгоритм;
b) з'ясувати, що дано і що треба знайти, розробити алгоритм;
c) на основі побудованої математичної моделі розробити програму;
d) на основі побудованої математичної моделі розробити блок–схему.

8. Розбивка алгоритму на окремі елементарні дії (команди), що легко виконуються даним виконавцем називається:
a) результативність;
b) дискретність;
c) масовість;
d) інваріантність

9. Що таке «компіляція програми» ?
a) створення exe–файла програми;
b) створення запускного файла програми;
c) активізація попереднього вікна;
d) меню для роботи з вікнами;
e) розгортання активного вікна на повний екран.

10. Як запустити програму на виконання?
a) Shift + Insert;
b) A1t + Enter;
c) F3;
d) пункт меню Run / Run;
e) Ctrl+F9.

11. Які із записаних процедур є процедурами введення даних ?

a) write(b);
b) writeln(' Привіт ');
c) readln(t) ;
d) read (m, n) ;
e) V:=7;

12. Вкажіть, де допущені помилки при виведенні даних, якщо b – змінна дійсного типу ?

a) writeln(‘ Значення рівне, b:2:4);
b) writeln(b:2:4) ;
c) write (‘ Значення рівне ‘, b:2:4);
d) writeln(b:2);
e)writeln(‘ Значення рівне ‘, b:2:4).

13. Як знайти квадратний корінь із суми квадратів x та y?
a) sqrt(x*x+y*y);
b) sqrt(sqr (x) + sqr(y)) ;
c) sqr(x*x+y*y);
d) sqr (sqrt(x) + sqrt(y)) ;
e) sqr (sqr (x) + y*y)) .

14. Якого значення набуде змінна y в результаті виконання дій: a: =5; if a>8 then y:=–1 else y: =10;?
a)–1;
b) 1;
c) 10;
d) 5;
e) 8.

15. Якого значення набуде змінна y в результаті виконання дій: a:=5; if (a>8) or (a<6) then y:=-1 else y:=10;?
a) -1;
b) 5;
c) 6;
d) 8;
e) 10.

16. Якого значення набуде змінна y в результаті виконання дій: a:=5; if (a>8) and (a<6) then y:=-1 else y:=10;?
a) -1;
b) 5;
c) 6;
d) 8;
e) 10.

*17. Якого значення набуде змінна m в результаті виконання дій: m:=1; for i: =2 to 5 do m:=m*i; ?*
a) 5;
b) 15;
c) 240;

d) 120;
e) 0.

18. Якого значення набуде змінна n в результаті виконання дій: n:=0; while n<10 do n:=n+3;?

a) 12;
b) 10;
c) 9;
d) 0;
e) 11.

19. Якого значення набуде змінна b в результаті виконання дій: b:=1; repeat b:=b/2; until b<=1;?

a) 1;
b) 0.5;
c) –0.5;
d) –1;
e) 0.

20. Змінна x: =3255. Яке буде отримано значення y в результаті виконання команди: y :=x div 100?

a) 32;
b) 32.55;
c) 0.3255;
d) 3.2E1;
e) 32.0.

21. Змінна x: =3255. Яке буде отримано значення y в результаті виконання команди: y:=x mod 100?

a) 55.0;
b) 55;
c) 0.55;
d) 3255E–2;
e) 32.55.

22. Де є помилки в записах функцій?

a) sin*3x;
b) ctn(2*x) ;
c) abs(2*x) ;
d) cos(3*x) ;
e) sin(3*x)^2.

23. Яка функція знаходить квадратний корінь виразу ?

a) sqr ;
b) sqrt ;
c) exp ;
d) abs ;
e) ln.

24. Як знайти тангенс змінної m?

a) tan(m) ;
b) sin/cos m;
c) sin(m)/cos(m) ;
d) tg (m) ;
e) tg m.

*25. a:=1; b:=–5; c:=-6. Якого значення набуде змінна d, якщо d:=sqrt(sqr(b)-4*a*c) ?*

a) 1;
b) 49;
c) -7;
d) -1;
e) 7.

РОЗДІЛ 2
СОРТУВАННЯ ЕЛЕМЕНТІВ У МАСИВАХ

План (логіка) викладу матеріалу:

В багатьох сферах діяльності, які пов’язані із записом, обробкою та збереженням інформації, при роботі з базами даних, у практиці програмування дуже часто виникає необхідність у впорядкуванні елементів використовуваних структур даних за якою–небудь ознакою, або, як іноді кажуть, за яким–небудь «ключем». Програмісти кажуть в цьому випадку, що необхідно відсортувати множину елементів, використовуючи задане відношення порядку [1, 22].

Незамінним інструментом сучасних дослідників є програми для наукових розрахунків. Розвиток обчислювальних методів зробив можливим розв’язання різноманітних по складності наукових завдань за допомогою обчислювальної техніки. Програмне забезпечення розробляється багатьма організаціями, як невеликими компаніями, так і великими корпораціями світового рівня. Випускаються спеціалізовані програми для самих різних дисциплін - математики, астрономії, хімії, фізики, біології, лінгвістики, інженерії, розробки штучного інтелекту тощо. Програмне забезпечення необхідне як при елементарному відтворенню графіків, так і при обробці великих масивів інформації,

зібраної науковими приладами [31].

Під ***сортуванням*** розуміють процес перестановки об'єктів даної множини в певному порядку [1, 6, 22].

Мета сортування – полегшення подальшого пошуку елементів у відсортованій множині (масиві). У цьому значенні елементи сортування присутні майже в усіх задачах.

У загальній постановці задача сортування ставиться таким чином. Є послідовність однотипних записів, одне з полів яких обране як ***ключ сортування***. Тип даних ключа повинен включати операції порівняння ***(«=», «>», «<», «>=»*** та ***«<=»).*** Задачею сортування є перетворення початкової послідовності у послідовність, що містить ті ж записи, але у порядку зростання (або убування) значень ключа. Метод сортування називається стійким, якщо при його застосуванні не змінюється відносне положення записів з рівними значеннями ключа.

Розрізняють сортування масивів записів, цілком розташованих в основній пам'яті ***(внутрішнє сортування)***, і сортування файлів, що зберігаються в зовнішній пам'яті і не поміщаються повністю в основній пам'яті ***(зовнішнє сортування)***. Для внутрішнього і зовнішнього сортування потрібні істотно різні методи. Тут розглядаються найвідоміші методи внутрішнього сортування, починаючи з простих і зрозумілих, але не дуже швидких, і закінчуючи ускладненими методами, що не так просто розуміються.

Природною умовою використання будь–якого методу внутрішнього сортування є те, що ці методи не повинні вимагати додаткової пам'яті: всі перестановки з метою впорядкування елементів масиву повинні виконуватися в межах того ж масиву. Мірою ефективності алгоритму внутрішнього сортування є число необхідних порівнянь значень ключа ***(C)*** і число перестановок (пересилок) елементів ***(M)*** [1]**.**

Основна вимога до методів сортування масивів – економне використання пам'яті. Це означає, що перепорядкування елементів потрібно виконувати на тому ж місці (з використанням мінімуму ресурсів пам'яті) що методи, які пересилають елементи з масиву ***A*** в масив ***B***, не представляють для нас інтересу. Проблема сортування неврегульованого масиву відноситься в обчислювальній техніці до класичних. Вона здається простою, адже всім доводилося виконувати яке–небудь механічне сортування, чи то розкладка гральних карт, гардеробних номерків, карток з бібліотечного каталога або грошових рахунків, але нажаль, простота ця ілюзорна.

Відмітимо, що оскільки сортування засноване тільки на значеннях ключів і ніяк не зачіпає поля записів, що залишилися, можна говорити про сортування масивів ключів.

Існує більше ніж 300 різновидів методів сортування [20-22]. Деякі

основні із методів внутрішнього сортуванням розглянемо детально.

1. Сортування обміном (методом «бульбашки»).
2. Бульбашкове сортування з просіванням.
3. Сортування вибором.
4. Сортування включеннями.
5. Бульбашкове сортування вставками.
6. Сортування методом знаходження мінімального елементу.
7. Сортування методом знаходження нового елементу.
8. Швидке сортування (сортування Хоара).
9. Сортування Шелла.
10. Шейкер-сортування.
11. Метод послідовного пошуку мінімумів.
12. Сортування деревом.
13. Пірамідальне сортування (турнірне сортування).
14. Сортування із злиттям.
15. Порозрядне цифрове сортування.
16. Модифікація бульбашкового сортування з ознакою.
17. Бульбашкове із запам’ятовуванням місця останньої перестановки.
18. Модифікація простими вставками.
19. Сортування підрахунком.
20. Оболонкове сортування.
21. Сортування підрахунком розподілень.

2.1. Сортування обміном

Сортування обміном – метод, при якому всі сусідні елементи масиву попарно порівнюються один з одним і міняються місцями в тому випадку, якщо попередній елемент більший ніж наступний. В результаті цього максимальний елемент поступово зміщується праворуч і врешті–решт займає крайнє праве місце в масиві, після чого він виключається з подальшої обробки [1, 22].

Потім процес повторюється і своє місце займає другий за величиною елемент, який також виключається з подальшого розгляду. Так продовжується до тих пір, доки вся послідовність не буде впорядкована. Сортування обміном називають ще ***«бульбашковим»*** (порівняння із спливанням міхурів повітря – «бульбашок» – в рідині; вважаємо, що масив розташований у стовпчик, та нагору «спливає», мов бульбашка, найлегший – найменший).

Просте обмінне сортування ***a[1], a[2], ..., a[n]*** працює таким чином. Починаючи з кінця масиву порівнюються два сусідні елементи ***(a[n] і a[n–***

1]). Якщо виконується умова ***a[n–1]> a[n],*** то значення елементів міняються місцями. Процес продовжується для ***a[n–1]*** та ***a[n–2]*** і т.д., доки не буде виконане порівняння ***a[2]*** та ***a[1].*** Після цього на місці ***a[1]*** опиняється елемент масиву з найменшим значенням. На другому кроці процес повторюється, але останніми порівнюються ***a[3]*** та ***a[2].*** І так далі. На останньому кроці порівнюватимуться тільки поточні значення ***a[n]*** та ***a[n–1].*** Зрозуміла аналогія з «бульбашкою», оскільки найменші елементи («найлегші») поступово «спливають» до верхньої межі масиву. Приклад сортування бульбашковим методом наведено у таблиці 2.1.

Метод сортування масиву бульбашковим методом найповільніший, але якщо масив невеликий і програма некритична до швидкості, то можна застосувати і його.

Таблиця 2.1

Приклад сортування бульбашковим методом

Початковий стан масиву	8 23 5 65 44 33 1 6
Крок 1	8 23 5 65 44 33 1 6 8 23 5 65 44 1 33 6 8 23 5 65 1 44 33 6 8 23 5 1 65 44 33 6 8 23 1 5 65 44 33 6 8 1 23 5 65 44 33 6 1 8 23 5 65 44 33 6
Крок 2	1 8 23 5 65 44 6 33 1 8 23 5 65 6 44 33 1 8 23 5 6 65 44 33 1 8 23 5 6 65 44 33 1 8 5 23 6 65 44 33 1 5 8 23 6 65 44 33
Крок 3	1 5 8 23 6 65 33 44 1 5 8 23 6 33 65 44 1 5 8 23 6 33 65 44 1 5 8 6 23 33 65 44 1 5 6 8 23 33 65 44
Крок 4	1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65
Крок 5	1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65
Крок 6	1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65
Крок 7	1 5 6 8 23 33 44 65

Алгоритм дуже простий (якщо два сусідні елементи розташовані не

по порядку, міняються їх місцями). Так повторюється до тих пір, доки в черговому проході не робиться жодного обміну, тобто масив буде впорядкованим. Якщо зафіксувати цей факт, то незначне ускладнення процедури сортування може дати істотний виграш у швидкості. Нижче наведено варіант, що враховує цю обставину.

Процедура сортування обміном:

```
PROCEDURE BubbleSort (var a: Integer);
var i,j:Integer;
 f:Boolean;{Якщо був обмін,то f=TRUE,ні – f=FALSE}
 BEGIN
  f := TRUE; i := N-1;
  While (i>=1) AND f do
  begin
   f:=FALSE;
   For j:=1 to i do If a[j]>a[j+1] then
   begin f := TRUE; Swap (a[j],a[j+1])
   End;
   i := i-1;
  end;
     END.
```

Необхідна також процедура, що забезпечує обмін місцями двох елементів – знайденого при ***i***–му перегляді мінімального елементу та елементу з індексом ***i*** (процедура перестановки).

Процедура перестановки:

```
  PROCEDURE Swap (var a,b: Integer);
  var c: Integer;
  BEGIN
   c := a; a := b; b := c
  END;
```

Алгоритм реалізації простого обміну:

```
 PROGRAM B_u_b_b_l_e_S_o_r_t (Input,Output);
  const N=8;
  type index = 1..N;
    massiv = Array[0..N] of Integer;
```

```
  var x : Integer;
    a : massiv;
    i,j: index;
  BEGIN
  For i:=1 to N do ReadLn (a[i]);
  For i:=2 to N do
   For j:=n downto i do
   If a[j-1]>a[j] then
   begin
    x:=a[j-1]; a[j-1]:=a[j]; a[j]:=x
   end;
  For i:=1 to N do Write (a[i],' ');
  END.
```

Для методу простого обмінного сортування потрібне число порівнянь ***(n(n–1)/2),*** мінімальне число пересилок ***0***, а середнє і максимальне число пересилок – $M(n^2)$.

2.2. Бульбашкове сортування з просіванням

Даний метод аналогічний методу бульбашкового сортування, але після перестановки пари сусідніх елементів виконується просівання: найменший лівий елемент просувається до початку масиву на скільки це можливо, поки не виконується умова впорядкованості.

Перевага: простий бульбашковий метод працює дуже повільно, а коли *мінімальний чи максимальний* елемент масиву (залежно від сортування) стоїть в кінці масиву, цей алгоритм набагато швидший.

```
Program BubbleSiftingSort;
const n = 10;
var

     x: array[1 .. n] of integer;
     i, j, t: integer;
     flagsort: boolean;

    procedure bubble_S_S;
    begin
     repeat
     flagsort := true;
     for i := 1 to n - 1 do
      if not(x[i]<=x[i + 1]) then begin
```

```
      t := x[i];
      x[i] := x[i + 1];
      x[i + 1] := t;
      j := i;
      while (j>1) and not(x[j-1]<=x[j]) do

      begin
       t:=x[j]; x[j]:=x[j-1]; x[j-1]:=t;
       dec(j);
      end;
      flagsort := false;
      end;
     until flagsort;
    end;

begin {Основна програма}
  …
  bubble_S_S;
  …
end;
```

Зауваження: алгоритм тестувався на масиві цілих чисел (25000 елементів), приріст швидкості щодо простого бульбашкового сортування склав близько 75% (тобто складність алгоритму бульбашкового сортування з просіванням приблизно ***(4/7)n***).

2.3. Сортування вибором

При цьому методі сортування у неврегульованій послідовності обирається мінімальний елемент, який виключається з подальшої обробки, а послідовність елементів, що залишилася, приймається за початкову. Процес повторюється до тих пір, доки всі елементи не будуть вибрані. Очевидно, що всі вибрані елементи утворюють впорядковану послідовність.

Обраний в початковій послідовності мінімальний елемент розміщується на призначеному йому місці впорядкованої послідовності декількома способами:

Спосіб 1. Мінімальний елемент після ***i***–го перегляду переміщається на ***i***–е місце (***i=1,2,3,...***) іншого спеціально створеного масиву, а в старому, початковому, на місці обраного розміщується якесь дуже велике число, що перевершує по величині будь–який елемент масиву, що сортується. Змінений таким чином масив приймається за початковий, і здійснюється наступний перегляд.

Спосіб 2. Мінімальний елемент після ***i***–го перегляду переміщається на ***i***–е місце (***i=1,2,3,...***) заданого масиву, а елемент з ***i***–го місця – на місце обраного. Після кожного перегляду впорядковані елементи (від першого до елементу з індексом ***i***) виключаються з подальшої обробки, тобто розмір кожного подальшого оброблюваного масиву на одиницю менше розміру попереднього.

Іншими словами при сортуванні масиву ***a[1], a[2], ..., a[n]*** методом простого вибору серед всіх елементів відшукується елемент з найменшим значенням ***a[i],*** потім ***a[1]*** та ***a[i]*** обмінюються значеннями. Цей процес повторюється для отриманих підмасивів ***a[2], a[3], ..., a[n], ... a[j], a[j+1], ..., a[n]*** до тих пір, доки ми не дійдемо до підмасиву ***a[n],*** що містить до цього моменту найбільше значення. Робота алгоритму ілюструється прикладом в таблиці 2.2.

Для методу сортування простим вибором необхідним є число порівнянь – ***(n(n–1)/2)***. Порядок необхідного числа пересилок (включаючи й ті, які потрібні для вибору мінімального елементу) у найгіршому випадку складає $O(n^2)$**.** Проте порядок середнього числа пересилок є ***O(n ln n)***, що у ряді випадків робить цей метод більш прийнятним.

Таблиця 2.2

Приклад сортування простим вибором

Початковий стан масиву	8 23 5 65 44 33 1 6
Крок 1	1 23 5 65 44 33 8 6
Крок 2	1 5 23 65 44 33 8 6
Крок 3	1 5 6 65 44 33 8 23
Крок 4	1 5 6 8 44 33 65 23
Крок 5	1 5 6 8 33 44 65 23
Крок 6	1 5 6 8 23 44 65 33
Крок 7	1 5 6 8 23 33 65 44
Крок 8	1 5 6 8 23 33 44 65

Програма сортування вибором потребує допоміжної функції пошуку індексу мінімального елементу із списку змінної довжини – від елементу з деяким індексом ***start*** до останнього елементу масиву ***(I_m_i_n),*** та також процедури, що забезпечує обмін місцями двох елементів – знайденого при ***i***–му перегляді мінімального елементу та елементу з індексом ***i*** (***Swap*** – див. вище)**.**

Функція пошуку індексу мінімального елементу із списку:

```
FUNCTION I_m_i_n(start:Integer; a:tarray):Integer;
  { Шукає мінімальний елемент праворуч від start
   (включаючи start) та повертає його номер}
var m: Integer;
 i: Integer;
BEGIN
  m := start;
  For i:=start+1 to N do
  If a[i]<a[m] then m := i;
  I_m_i_n :=m
END;
```

Процедура сортування вибором:

```
    PROCEDURE sort_choice (var t:array);
    var i: Integer;
    BEGIN
     For i:=1 to N-1 do Swap(a[I_m_i_n(i,a)],a[i])
    END;
```

Програма алгоритму простого вибору:

```
PROGRAM Sort(Input,Output);
       {Сортування простим вибором}
 const N=8;
 type index = 0..N;
   massiv = Array[0..N] of Integer;
 var x : Integer;
  a : massiv;
  i,j,k: index;
 BEGIN
  For i:=1 to N do readln(a[i]);
  For i:=1 to N-1 do
  begin
    k := i; x := a[i];
    For j:=i+1 to N do
    If a[j]<x then
    begin
```

```
    k := j; x := a[j]
   end;
   a[k] := a[i]; a[i] := x
  end;
  For i:=1 to N do Write (a[i],' ')
END.
```

2.4. Сортування включеннями

Одним з найпростіших і природніх методів внутрішнього сортування є ***сортування з простими включеннями***. Ідея алгоритму дуже проста. Існує масив ключів ***a[1], a[2], ..., a[n].*** Для кожного елементу масиву, починаючи з другого, виконується порівняння з елементами з меншим індексом (елемент ***a[i]*** послідовно порівнюється з елементами ***a[i–1], a[i–2]...***) і до тих пір, доки для чергового елементу ***a[j]*** виконується співвідношення ***a[j]> a[i], a[i]*** та ***a[j]*** міняються місцями. Якщо вдається зустріти такий елемент ***a[j],*** що ***a[j]<=a[i]***, або якщо досягнута нижня межа масиву, виконується перехід до обробки елементу ***a[i+1]*** (доки не буде досягнута верхня межа масиву).

Приклад сортування методом простого включення представлено у таблиці 2.3.

Таблиця 2.3

Приклад сортування методом простого включення

Початковий стан масиву	8 23 5 65 44 33 1 6
Крок 1	8 23 5 65 44 33 1 6
Крок 2	8 5 23 65 44 33 1 6 5 8 23 65 44 33 1 6
Крок 3	5 8 23 65 44 33 1 6
Крок 4	5 8 23 44 65 33 1 6
Крок 5	5 8 23 44 33 65 1 6 5 8 23 33 44 65 1 6
Крок 6	5 8 23 33 44 1 65 6 5 8 23 33 1 44 65 6 5 8 23 1 33 44 65 6 5 8 1 23 33 44 65 6 5 1 8 23 33 44 65 6 1 5 8 23 33 44 65 6
Крок 7	1 5 8 23 33 44 6 65 1 5 8 23 33 6 44 65 1 5 8 23 6 33 44 65 1 5 8 6 23 33 44 65 1 5 6 8 23 33 44 65

У найкращому випадку (коли масив вже впорядкований) для виконання алгоритму з масивом із ***n*** елементів потрібно ***n–1*** порівняння і ***0*** пересилок. У найгіршому випадку (коли масив впорядкований в зворотному порядку) потрібно ***(n(n–1)/2)*** порівнянь і стільки ж пересилок. Таким чином, можна оцінювати складність методу простих включень як $O(n^2)$.

При сортуванні включеннями з неврегульованої послідовності елементів по черзі обирається кожен елемент, порівнюється з попереднім (вже впорядкованим) списком і поміщається на відповідне місце в останньому. Для сортування включеннями необхідна допоміжна процедура, яка забезпечує переміщення елемента з індексом ***i*** на місце елементу з індексом ***j*** та зсув усіх елементів з індексами від ***j*** до ***i*** на одну позицію праворуч.

Програма реалізації алгоритму сортування простим включеннями:

```
PROGRAM S_o_r_t(Input,Output);
    {Сортування простим включенням}
const N=8;
type index=0..N;
  massiv = Array[0..N] of Integer;
var x: Integer;
  a: massiv;
 i,j: index;
BEGIN
 For i:=1 to N do ReadLn (a[i]);
 For i:=2 to N do
  begin
   x := a[i]; a[0] := x; j := i-1;
   While x<a[j] do
    begin a[j+1] := a[j]; j := j-1 end;
   a[j+1] := x
  end;
 For i:=1 to N do Write (a[i],' ')
 END.
```

Програма реалізації алгоритму сортування бінарним включеннями:

```
PROGRAM Sort (Input,Output);
      {Сортування бінарним включенням}
 const N=8;
 type index = 0..N;
```

```
 massiv = Array[0..N] of Integer;
var x  : Integer;
 a  : massiv;
 i,j,l,r,m: index;
BEGIN
 For i:=1 to N do Readln (a[i]);
 For i:=2 to N do
 begin
  x := a[i]; l := 1; r := i-1;
  While l<=r do
  begin
  m := (l+r) DIV 2;
  If x<a[m] then r:=m-1 else l:= m+1
  end;
  For j:=i-1 downto l do a[j+1] := a[j];
  a[l] := x;
 end;
 For i:=1 to N do Write (a[i],' ')
END.
```

Можливо скоротити число порівнянь у методі простих включень, якщо скористатися тим фактом, що при обробці елементу ***a[i]*** масиву елементи ***a[1], a[2], ..., a[i–1]*** вже впорядковані, і скористатися для пошуку елементу, з яким повинна бути виконана перестановка, методом двійкового ділення.

В цьому випадку оцінка числа необхідних порівнянь стає ***O (n log n)***. Оскільки при виконанні перестановки потрібне зсовування на один елемент декількох елементів, то оцінка числа пересилок залишається $O(n^2)$.

2.5. Бульбашкове сортування вставками

Ще більш швидкий і оптимальний метод сортування – ***сортування вставками.*** Суть методу у тому, що на ***n***–ному кроці ми маємо впорядковану частину масиву з ***n*** елементів, і наступний елемент встає на відповідне йому місце. Перший індекс масиву – ***0***.

Програма бульбашкового сортування масиву вставками:

```
procedure SortInsert (var Arr : array of Integer;
       n : Integer);
var i, j, Temp : Integer;
```

```
begin
 for i := 1 to n do begin
 Temp := Arr [i]; j := i - 1;
 while Temp < Arr [j] do begin
  Arr [j + 1] := Arr [j]; Dec (j);
  if j < 0 then Break;
 end;
 Arr [j + 1] := Temp;
 end;
end;
```

2.6. Сортування методом знаходження мінімального елементу

Цей метод сортування швидший, ніж бульбашковий метод. Полягає він в наступному: при кожному перегляді масиву знаходимо мінімальний елемент і міняємо місцями його з першим на першому проході, з другим – на другому і т.д. Перший елемент масиву повинен мати індекс ***0***.

Програма сортування методом знаходження мінімального елементу:

```
procedure SortMin (var Arr : array of Integer;
       n : Integer);
var   i, j : Integer;
  Min, Pos, Temp : Integer;
begin
 for i := 0 to n - 1 do
 begin
 Min := Arr [i];
 Pos := i;
 for j := i + 1 to n do
  if Arr [j] < Min then
  begin
  Min := Arr [j];
  Pos := j;
 end;
 Temp := Arr [i];
 Arr [i] := Arr [Pos];
 Arr [Pos] := Temp;
 end;
```

```
end.
```

2.7. Сортування методом знаходження нового елементу

Сортування методом знаходження нового елементу (нового номеру) виконується у новий додатковий масив. Послідовно для кожного елементу масиву обчислюється його нова позиція у новому відсортованому масиві, розраховується кількість елементів, значення яких:

1) < значення елементу, який аналізується;

2) значення яких = значенню елементу, який аналізується та номера яких <= номеру елементу, який аналізується .

Особливість методу полягає у необхідності додаткового масиву, який не чутливий до початкової впорядкованості.

```
Type TArr = array[1..100] of integer;
Var mass1, NewMass : TArr;
     n: integer;
{n – розмірність масиву, mass1 – вхідний масив,
 NewMass – буде містити відсортовані елементи mass1}
procedure NewNSort(var mass, Nmass: TArr; size:
integer);
var i, j, NewN: integer;
begin
 for i := 1 to size do begin
 NewN := 0;
 for j := 1 to size do
  if (mass[j]<mass[i]) or ((mass[j]=mass[i]) and
(j<=i)) then inc(NewN);
 Nmass[NewN] := mass[i];
 end;
end;
```

Виклик процедури: ***NewNSort(mass1,NewMass,n);***

Масив ***NewMass*** буде містити відсортовані елементи масиву **mass1**. Метод добре працює з невеликими масивами, але із збільшенням розміру масиву «метод нового номери» починає значно програвати порозрядному сортуванню. Алгоритмічна складність складає у найкращому випадку

O(n), а у найгіршому ***O(n^2).***

2.8. Швидке сортування (сортування Хоара)

Сортування Чарльза Хоара (C.Hoare – Хор, Хоор, Хоар), або алгоритм ***швидкого сортування Хоара*** це удосконалений метод сортування, що базується на сортуванні обмінами. Тобто, спочатку ми обираємо деякий «бар'єр» (один з елементів масиву). Потім ми переглядаємо елементи, що стоять зліва від «бар'єра» і порівнюємо їх з ним. Коли ми знаходимо елемент, який є більшим за «бар'єр», то ми починаємо переглядати масив з кінця, порівнюючи елементи з «бар'єром». Якщо ми знайдемо в правій частині масиву елемент менший за «бар'єр», то ми переставимо місцями елемент зліва (той, що більше за «бар'єр») і елемент з права (той, що менше) і продовжимо перегляд. Потім ми рекурсивно застосовуємо цю процедуру для того, щоб відсортувати початок і кінець масиву.

По суті алгоритм Хоара працює на основі алгоритму сортування обмінами, але цей алгоритм вважається швидким, оскільки перегляд послідовності відбувається у ***двох напрямках одночасно***.

Ідея Ч. Хоара полягає в наступному:

1. Обираємо деякий елемент ***x*** масиву ***A*** випадковим образом.
2. Переглядаємо масив у прямому напрямку ***(i = 1, 2,...),*** шукаючи в ньому елемент ***A[i]*** не менший за ***x.***
3. Переглядаємо масив у зворотньому напрямку ***(j = n, n–1,..),*** шукаючи в ньому елемент ***A[j]*** не більший за ***x.***
4. Змінюємо місцями ***A[i]*** та ***A[j].***
5. Пункти 2–4 повторюємо доти, доки **i < j.**

У результаті такого зустрічного проходу початок масиву ***A[1..i]*** і кінець масиву ***A[j..n]*** виявляються розділеними «бар'єром» ***x: A[k] < x*** при ***k < i*** та ***A[k] < x*** при ***k > j*** , причому на поділ ми витратимо не більш ***n/2*** перестановок. Тепер залишилося проробити ті ж дії з початком і кінцем масиву, тобто застосувати їх рекурсивно.

Таким чином, процедура ***Hoare*** (рекурсивний алгоритм) залежить від параметрів ***k*** та ***m*** – початкового і кінцевого індексів відрізка масиву, який обробляється.

Процедура Hoare (рекурсивний алгоритм швидкого сортування):

```
Procedure Hoare(L, R : Integer);
  Var left, right : Integer;
```

```
        x : Integer;

   Begin
    If L < R then
    begin
     x := A[(L + R) div 2];  {вибір бар'єра x}
     left := L; right := R ;
     Repeat      {зустрічний прохід}
     While A[left]<x do Inc(left);
            {перегляд уперед}
     While A[right]>x do Dec(right);
            {перегляд назад}
     If left <= right then

     begin
      Swap(left, right);  {перестановка}
     Inc(left); Dec(right);
     end
    until left > right;
    Hoare(L, right);   {сортуємо початок}
    Hoare(left, R)    {сортуємо кінець}
    End;
```

Процедура алгоритму швидкого сортування Хоара за допомогою циклів While, Repeat Until та оператору If, який відповідає за порівняння (рекурсивний алгоритм):

```
   Procedure HoareSearch ( var a:mas; L, R: Integer);
     Var left, right, b, x: Integer;

   Begin
     if L < R then
     begin
     x:= a[( L+R) div 2];
     left:= L;
     right:= R;
     Repeat
      While a[ left] < x do left:= Succ(left);
      While a[right] >x do right:= Pred(right);
      If left >= right then
      Begin
      b:= a[left];
```

```
      a[left]:= a[right];
      a[right]:=b;
      left:= Succ( left);
      right:= Pred(right);
      End;
    Until left > right;
    HoareSearch ( L, right);
    HoareSearch (left, R);
    end;
  End;
```

Іншими словами під час роботи алгоритму швидкого сортування відбувається аналіз даної послідовності одночасно у двох напрямках (зліва–направо і справа–наліво). Комп'ютер порівнює два елементи, що стоять поряд зліва. Якщо ці елементи стоять на своїх місцях, тобто перший з них є меншим за другий, то комп'ютер порівнює перший елемент з останнім. Якщо при порівнянні останній елемент виявиться меншим за перший, то комп'ютер виконає перестановку цих двох елементів. Такі дії будуть відбуватися до тих пір, поки індикатор, якій відповідає за ліву частину послідовності (в процедурі ***Qsort – «i»***) не перейде на праву частину, а індикатор, що відповідає за праву частину масиву (в процедурі ***Qsort – «j»***) не перейде на праву частину. Далі та ж сама процедура викликається рекурсивно. Тобто, якщо ліва частина вже відсортована, то ми викликаємо ту саму процедуру і комп'ютер виконує ті самі дії, але в параметрах процедури ми змінюємо ліву границю. Те саме відбувається, коли відсортована права частина масиву.

Процедуру ***Qsort*** реалізує швидке сортування масиву, що вводиться за алгоритма Хоара з використанням оператору ***if***, який відповідає за порівняння елементів, та пересилання.

У таблиці 2.4 надано приклад сортування Хоара.

Таблиця 2.4

Приклад сортування Хоара

Початковий стан масиву	8 23 5 65 |44| 33 1 6
Крок 1 за ***x*** обирається ***a[5]***	├──────────┤ 8 23 5 6 44 33 1 65 ├──┤ 8 23 5 6 1 33 44 65
Крок 2 в підмасиві ***a[1]***, ***a[5]*** за ***x*** обирається ***a[3]***	8 23 |5| 6 1 33 44 65 ├──────────┤ 1 23 5 6 8 33 44 65 ├──┤ 1 5 23 6 8 33 44 65

Крок 3 в підмасиві ***a[3], a[5]*** за ***x*** обирається ***a[4]***	1 5 23 |6| 8 33 44 65 |———| 1 5 8 6 23 33 44 65
Крок 4 в підмасиві ***a[3], a[4]*** за ***x*** обирається ***a[4]***	1 5 8 |6| 23 33 44 65 |———| 1 5 6 8 23 33 44 65

Процедура алгоритму швидкого сортування Хоара за допомогою оператора If (рекурсивний алгоритм):

```
Procedure Qsort (var a:mas; low, hi: byte);
  Var i,j:byte;

begin
  if hi> low then
  begin
  i:= low;
  j:= hi;
  x:= a[i];
  While i< j do if a[i+1]<=x then
  begin
   a[i]:= a[i+1];
  a[i+1]:=x;
  i:= i+1;
  end
  else
  begin
  if a[j]<=x then
  begin
   y:=a[j];
   a[j]:=a[i+1];
   a[i+1]:=y;
  end;
  j:=j-1
  end;
  Qsort (a, low, i-1);
  Qsort (a, i+1, hi);
 end;
```

Програма швидкого сортування (нерекурсивний алгоритм):

```
PROGRAM NonRecursiveQuickSort (Input,Output);
```

```
 {Увага!!! В програмі використаний тип Record}
const N = 8; M = 12;
type index = 0..N;
 massiv = Array [0..N] of Integer;
 zapis = Record;
  L,R : index;
var x,w : Integer;
 a  : massiv;
 i,j,L,R: index;
 s  : 0..M;
 stack : Array [1..M] of zapis;
BEGIN
 For i:=1 to N do ReadLn (a[i]);
 s := 1; stack[1].L := 1; stack[s].R := N;
 Repeat {Вибір із стека останнього запиту}
  L:= stack[s].L; R := stack[s].R; s := s-1;
  Repeat   {Розділення a[L],...,a[R]}
   i := L; j := R; x := a[(L+R) DIV 2];
   Repeat
    While a[i]<x do i := i+1;
    While x<a[j] do j := j-1;
    If i<=j then
    begin
    w:=a[i]; a[i]:=a[j];
    a[j]:= w; i:=i+1; j:=j-1;
    end;
   until i>j;
   If i<R then
   {Запис в стек запиту з правої частини}
   begin
   s:=s+1; stack[s].L:=i; stack[s].R:=R;
   end;
   R:=j
              {Тепер L і R обмежують ліву частину}
  until L>=R;
 until s=0;
 For i:=1 to N do Write (a[i],' ')
 END.
```

Алгоритм Хоара недаремно називається швидким сортуванням,

оскільки для нього оцінкою числа порівнянь і обмінів є ***O(n log n).*** Та й насправді, у більшості утиліт, що виконують сортування масивів, використовується саме цей алгоритм.

Приклад використання сортування Хоара

Пошук медіани в масиві. Медіаною послідовності з ***N*** елементів називається елемент, значення якого менше (або дорівнює) половини ***N*** елементів і більше (або дорівнює) іншій половині. Наприклад, медіаною масиву ***16 22 99 95 18 87 10*** є елемент ***18***.

Задачу пошуку медіани прийнято пов'язувати з сортуванням, оскільки медіану завжди можна знайти таким чином: відсортувати ***N*** елементів і потім вибрати середній елемент. Наведена нижче програма використовує алгоритм Хоара та дозволяє потенційно знайти медіану значно швидше (за час, значно менший, ніж час на повне сортування масиву).

Програма пошуку медіани масиву з використанням алгоритму Хоара:

```
PROGRAM M_e_d_i_a_n_a (Input,Output);
  const N=7;
  type index = 0..N;
  massiv = Array [0..N] of Integer;
  var x : Integer;
    a : massiv;
    i,k: index;
 { -------------- }
 PROCEDURE F_i_n_d (k: Integer);
  var l,r,i,j,w,x: Integer;
  BEGIN
  l:=1; r:=n;
  While l<r do
   begin
    x:=a[k]; i:=l; j:=r;
    Repeat
     While a[i]<x do i:=i+1;
     While x<a[j] do j:=j-1;
     If i<=j
     then begin
        w:=a[i]; a[i]:=a[j];
        a[j]:=w; i:=i+1; j:=j-1
       end
```

```
    until i>j;
    If j<k then l:=i;
    If k<i then r:=j
   end
 END;
{ --------------- }
 BEGIN
  For i:=1 to N do
  begin
            a[i]:=Random(23); Write(a[i],' ');
     end;
  WriteLn;
        ReadLn (k); F_i_n_d(k);
        WriteLn (a[k])
 END.
```

2.9. Сортування Шелла

Сортування методом Шелла, або ***сортуванням включеннями з відстанню, що зменшується*** є подальшим розвитком методу сортування з включеннями. Розглянемо випадок, коли число елементів в масиві сортування є ступенем числа ***2.*** Для масиву з ***2n*** елементами алгоритм працює таким чином. На першій фазі виконується сортування включенням всіх пар елементів масиву, відстань між якими є ***2(n–1)***. На другій фазі виконується сортування включенням елементів одержаного масиву, відстань між якими є ***2(n–2).*** І так далі, доки ми не дійдемо до фази з відстанню між елементами, що дорівнює одиниці, і не виконаємо завершальне сортування з включеннями. Приклад використання методу Шелла до масиву наведено в таблиці 2.5.

Таблиця 2.5

Приклад використання методу сортування Шелла

Початковий стан масиву	8 23 5 65 44 33 1 6
Фаза 1 (сортуються елементи, відстань між якими чотири)	8 23 5 65 44 33 1 6 8 23 5 65 44 33 1 6 8 23 1 65 44 33 5 6 8 23 1 6 44 33 5 65

Фаза 2 (сортуються елементи, відстань між якими два)	1 23 8 6 44 33 5 65 1 23 8 6 44 33 5 65 1 23 8 6 5 33 44 65 1 23 5 6 8 33 44 65 1 6 5 23 8 33 44 65 1 6 5 23 8 33 44 65 1 6 5 23 8 33 44 65
Фаза 3 (сортуються елементи, відстань між якими один)	1 6 5 23 8 33 44 65 1 5 6 23 8 33 44 65 1 5 6 23 8 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65

У загальному випадку алгоритм Шелла визначається для заданої послідовності із ***t*** відстаней між елементами ***h1, h2, ..., ht***, для яких виконуються умови ***h1 = 1 і h(i+1)< hi***. Дональд Кнут показав, що при правильно підібраних ***t*** та ***h*** складність алгоритму Шелла є $O(n^{1.2})$, що істотно менше за складність простих алгоритмів сортування.

Програма сортування Шелла:

```
PROGRAM S_h_e_l_l_s_o_r_t (Input,Output);
  const N = 780;
    t = 4;
  type Index = 0..N;
    Massiv = Array [-9..N] of Integer;
    Mas = Array [1..t] of Integer;
  var x  : Integer;
    a  : Massiv;
    h  : Mas;
    i,j,k,s: Index;
    m  : 1..t;
    L  : Index;

  BEGIN
  Writeln ('Введіть початковий масив...');
  For L:=1 to N do
  begin
   a[L] := Random (23);Write (a[L],' ');
  end;
```

```
WriteLn;
h[1] := 9; h[2] := 5; h[3] := 3; h[4] := 1;
For m:=t downto 1 do

begin
 k := h[m]; s := -k; {Місце бар'єру}
 For i:=k+1 to n do
 begin
 x := a[i]; j := i-k;
 If s=0 then s := -k;
 s := s+1; a[s] := x;
 While x<a[j] do

 begin
  a[j+k] := a[j]; j := j-k;
 end;
 a[j+k] := x
 end;
end;
Writeln ('Результати сортування...');
For L:=1 to N do Write (a[L],' ');
WriteLn
END.
```

2.10. Шейкер–сортування

Метод шейкер–сортування (шейкерного сортування – ***ShakerSort***) є вдосконаленням бульбашкового методу сортування за трьома простими ознаками:

1) якщо на деякому кроці не було виконано жодного обміну, то виконання алгоритму можна припиняти;

2) можна запам'ятовувати найменше значення індексу масиву, для якого на поточному кроці виконувалися перестановки; очевидно, що верхня частина масиву до елементу з цим індексом вже відсортована і на наступному кроці можна припиняти порівняння значень сусідніх елементів досягнувши такого значення індексу;

3) метод «бульбашки» працює нерівноправно для «легких» і «важких» значень; легке значення потрапляє на потрібне місце за один крок, а важке на кожному кроці опускається у напрямку до потрібного місця на одну позицію.

При застосуванні шейкер–сортування на кожному наступному кроці

міняється напрямок послідовного перегляду масиву. В результаті на одному кроці «спливає» черговий найлегший елемент, а на іншому «тоне» черговий найважчий. Приклад шейкер–сортування наведено в табл. 2.6.

Таблиця 2.6

Приклад шейкер-сортування

Початковий стан масиву	8 23 5 65 44 33 1 6
Крок 1	8 23 5 65 44 33 1 6 8 23 5 65 44 1 33 6 8 23 5 65 1 44 33 6 8 23 5 1 65 44 33 6 8 23 1 5 65 44 33 6 8 1 23 5 65 44 33 6 1 8 23 5 65 44 33 6
Крок 2	1 8 23 5 65 44 33 6 1 8 5 23 65 44 33 6 1 8 5 23 65 44 33 6 1 8 5 23 44 65 33 6 1 8 5 23 44 33 65 6 1 8 5 23 44 33 6 65
Крок 3	1 8 5 23 44 6 33 65 1 8 5 23 6 44 33 65 1 8 5 6 23 44 33 65 1 8 5 6 23 44 33 65 1 5 8 6 23 44 33 65
Крок 4	1 5 6 8 23 44 33 65 1 5 6 8 23 44 33 65 1 5 6 8 23 44 33 65 1 5 6 8 23 33 44 65
Крок 5	1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65

Шейкер-сортування дозволяє скоротити число порівнянь (за оцінкою ***Кнута*** середнє число порівнянь складає $(n^2 - n(const + \ln n))$, хоча порядком оцінки як і раніше залишається n^2. Число ж пересилок, взагалі не змінюється. Шейкер-сортування рекомендується використовувати у тих випадках, коли відомо, що масив «майже впорядкований».

Програма алгоритму шейкер–сортування:

```
PROGRAM S_h_a_k_e_S_o_r_t (Input,Output);
  const N=8;
```

```
type index = 0..N;
  massiv = Array[0..N] of Integer;
var x:   Integer;
  a:   massiv;
  i,j,k,l,r: index;

BEGIN
For i:=1 to N do ReadLn (a[i]);
l := 2; r := n; k := n;
Repeat
 For j:=r downto l do
  If a[j-1]>a[j] then
  begin
  x:=a[j-1];
  a[j-1]:=a[j];
  a[j]:=x;
  k:= j;
  end;

 l := k + 1;
 For j:=l to r do
  If a[j-1]>a[j] then
  begin
  x:=a[j-1];
  a[j-1]:=a[j];
  a[j]:=x;
  k:=j;
  end;
 r := k - 1;
until l>r;
For i:=1 to N do Write (a[i],' ');
   END.
```

2.11. Метод послідовного пошуку мінімумів

Переглядається увесь масив, шукається мінімальний елемент і ставиться на місце першого, «старий» перший елемент ставиться на місце знайденого.

Приклад методу послідовного пошуку мінімумів за допомогою процедури:

```
type TArr = array[1 .. 100] of integer;
var mass1: TArr;
   n : integer;
procedure NextMinSearchSort(var mass: TArr;
         size: integer);
var i, j, Nmin, temp: integer;
begin
 for i := 1 to size - 1 do begin
 nmin := i; for j := i + 1 to size do
 if mass[j]<mass[Nmin] then Nmin:=j;temp:=mass[i];
 mass[i] := mass[Nmin]; mass[Nmin] := temp;
 end;
end;
```

Виклик: ***NextMinSearchSort(mass1, n);***

2.12. Сортування деревом

Алгоритм сортування деревом (***TreeSort)*** є поліпшенням алгоритму сортування вибором. Процедура вибору найменшого елемента удосконалена як процедура побудови так званого сортуючого дерева. Сортуюче дерево – це структура даних, у якій представлений процес пошуку найменшого елемента методом попарного порівняння елементів, що стоять поруч. Алгоритм сортує масив у два етапи:

1) етап – побудова сортуючого дерева;

2) етап – просівання елементів по сортуючому дереву.

Розглянемо приклад: Нехай масив ***A*** складається з ***8*** елементів (рис. 2.1. 1-й рядок). Другий рядок складається з мінімумів елементів першого рядка, які стоять поруч. Кожний наступний рядок складений з мінімумів елементів, що стоять поруч, попереднього рядка.

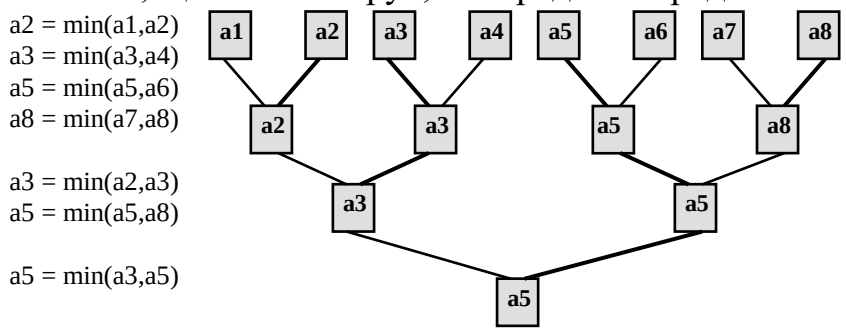


Рис. 2.1. Приклад сортування деревом (початок)

Ця структура даних називається сортуючим деревом. У корені сортуючого дерева розташований найменший елемент. Дерево упорядкованого масиву відповідає наступним властивостям: ***A[i] < =A[2i],***

A[i] < =A[2i+1], A[2i] < =A[2i+1]. Крім того, у дереві побудовані шляхи елементів масиву від листів до відповідного величині елемента вузла – розгалуження (рис. 2.1) шлях мінімального елемента ***a5*** – від листа ***a5*** до кореня відзначений товстою лінією). Коли дерево побудоване, починається етап просівання елементів масиву по дереву. Мінімальний елемент пересилається у вихідний масив ***B*** і усі входження цього елемента в дереві замінюються на спеціальний символ ***M*** (рис.2.2).

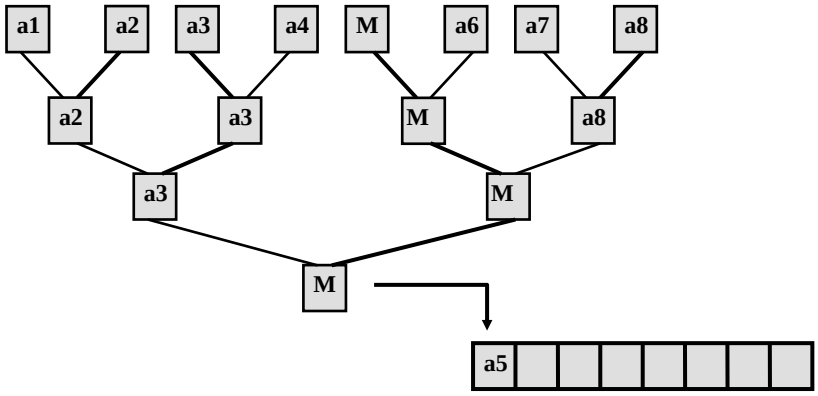


Рис.2.2. Приклад сортування деревом
(мінімальний елемент пересилається у вихідний масив ***B***)

Потім здійснюється просівання елемента уздовж шляху, відзначеного символом ***M***, починаючи з листка, сусіднього з ***M*** до кореня. Крок просівання – це вибір найменшого з двох елементів, що зустрілися на шляху до кореня дерева і його пересилання у вузол, відзначений ***M***. Просівання ***2***–го елемента показано на рис. 2.3 (символ ***M*** більше, ніж будь–який елемент масиву).

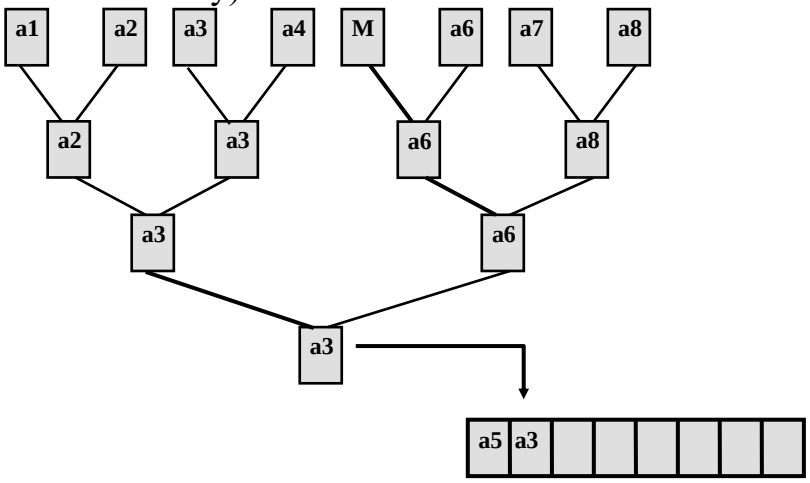


Рис. 2.3. Приклад сортування деревом (просівання ***2***–го елемента)

При цьому,

a6 = min(M, a6); a6 = min(a6, a8); a3 = min(a3, a6); b2 := a3

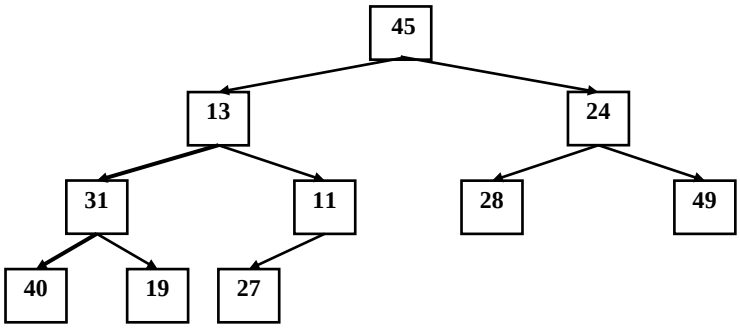
Просівання елементів відбувається доти, доки весь вихідний масив не

буде заповнений символами ***M***, тобто ***n*** раз:

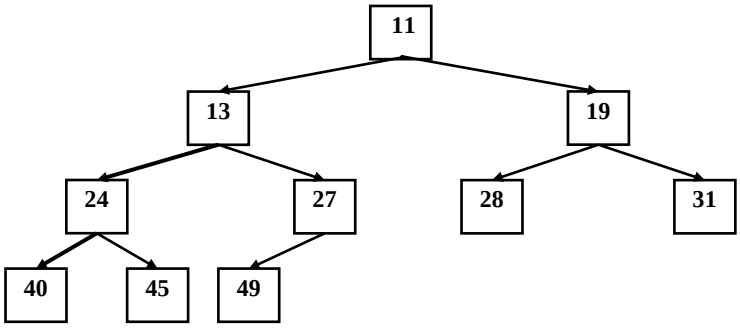
```
For I := 1 to n do
begin
 Відзначити шлях від кореня до листка символом M;
 Просіяти елемент уздовж відзначеного шляху;
    B[I]:= корінь дерева;
end;
```

Сортуюче дерево можна реалізувати, використовуючи або двовимірний масив, або одномірний масиві ***ST[1..N],*** де ***N = 2n–1***. Оцінимо складність алгоритму Насамперед відзначимо, що алгоритм ***TreeSort*** працює однаково на усіх входах, так що його складність у гіршому випадку збігається зі складністю в середньому.

Приклад. Задано масив ***A = [45 13 24 31 11 28 49 40 19 27]*** . На рис. 2.4 наведені відповідні йому дерева.



а)



б)

а) вхідного масиву ***A = [45 13 24 31 11 28 49 40 19 27]*** ;
б) вихідного масиву ***A = [11 13 19 24 27 28 31 40 45 49]*** .

Рис. 2.4. Приклад дерева

Оцінки складності алгоритму ***TreeSort*** в термінах ***N(n), C(n)*** за часом складає:

$$N(n) = O(n \log_2 n),\ C(n) = O(n \log_2 n).$$

У загальному випадку, коли ***n*** не є ступенем ***2***, сортуюче дерево будується трохи інакше. «Зайвий» елемент (елемент, для якого немає пари) переноситься на наступний рівень. Легко бачити, що при цьому глибина сортуючого дерева дорівнює $[\log_2 n] + 1$. Удосконалення алгоритму II етапу очевидно. Оцінки при цьому змінюють лише мультиплікативні множники. Алгоритм ***TreeSort*** має істотний недолік: для нього потрібно додаткова пам'ять розміру $2n - 1$.

Приклад програми сортування за методом дерева:

```
Program TreeSort;
Const n = 8;
Type TType = Integer;
  arrType = Array[1 .. n] Of TType;
Const a: arrType = (44, 55, 12, 42, 94, 18, 6, 67);

  { Сортування за допомогою бінарного дерева }
Type
 PTTree = ^TTree;
 TTree = Record
 a: TType;
 left, right: PTTree;
 End;
   { Додавання чергового елемента у дерево }
Function AddToTree(root:PTTree;nValue:TType):PTTree;
Begin
 {При відсутності наступника створити новий елемент}
 If root = nil Then Begin
 root := New(PTTree);
 root^.a := nValue;
 root^.left := nil;
 root^.right := nil;
 AddToTree := root; Exit
 End;
 If root^.a < nValue Then
 root^.right := AddToTree(root^.right, nValue)
 Else
 root^.left := AddToTree(root^.left, nValue);
 AddToTree := root
End;
       { Заповнення масиву }
```

```
Procedure TreeToArray(root: PTTree; Var a: arrType);
Const maxTwo: Integer = 1;
Begin
{ При відсутності наступників рекурсія зупиниться }
 If root = nil Then Exit;  { Ліве піддерево }
 TreeToArray(root^.left, a);
 a[maxTwo] := root^.a; Inc(maxTwo);
        { Праве піддерево }
 TreeToArray(root^.right, a);
 Dispose(root)
End;
     { Власне процедура сортування }
Procedure SortTree(Var a: arrType; n: Integer);
Var
 root: PTTree;
 i: Integer;
Begin
 root := nil;
 For i := 1 To n Do
 root := AddToTree(root, a[i]);
 TreeToArray(root, a)
End;

Var i: Integer;
Begin
 WriteLn('До сортування:');
 For i := 1 To n Do Write(a[i]:4);
 WriteLn;
 SortTree(a, n);
 WriteLn('Після сортування:');
 For i := 1 To n Do Write(a[i]:4);
 WriteLn
End.
```

Основний ***недолік*** цього методу – великі вимоги до пам'яті під дерево. Очевидно, потрібно ***n*** місця під ключі і, крім того, пам'ять на ***2*** покажчики для кожного з них.

Тому ***TreeSort*** звичайно застосовують там, де:

1) побудоване дерево можна з успіхом застосувати для інших задач;
2) дані вже побудовані в «дерево»;
3) дані можна прочитувати безпосередньо в дерево, наприклад, при

потоковому введенні з консолі або з файлу. Тобто там, де не вимагається додаткової пам’яті.

2.13. Пірамідальне сортування (турнірне сортування)

Пірамідальне сортування ***(Heapsort)*** є більш досконалим алгоритмом у порівнянні з сортуванням деревом та також використовує представлення масиву у вигляді дерева. Алгоритм пірамідального сортування (***HeapSort)*** не вимагає допоміжних масивів, сортуючи «на місці». Його ідея полягає у тому, що замість повного дерева порівняння початковий масив ***a[1], a[2], ..., a[n]*** перетворюється у піраміду, з тією властивістю, що для кожного ***a[i]*** виконуються умови ***a[i]<= a[2i]*** та ***a[i]<= a[2i+1].*** Потім піраміда використовується для сортування.

Найбільш наочний метод побудови піраміди використовує деревовидне представлення масиву ***(8,23,5,65,44,33,1,6)*** (рис. 2.5). Масив представляється у вигляді двійкового дерева, коріння якого відповідає елементу масиву ***a[1].*** На другому ярусі знаходяться елементи ***a[2]*** та ***a[3].*** На третьому – ***a[4], a[5], a[6], a[7]*** і т.д. Як видно, для масиву з непарною кількістю елементів відповідне дерево буде збалансованим, а для масиву з парною кількістю елементів ***n*** елемент ***a[n]*** буде єдиним (найлівішим) листом «майже» збалансованого дерева.

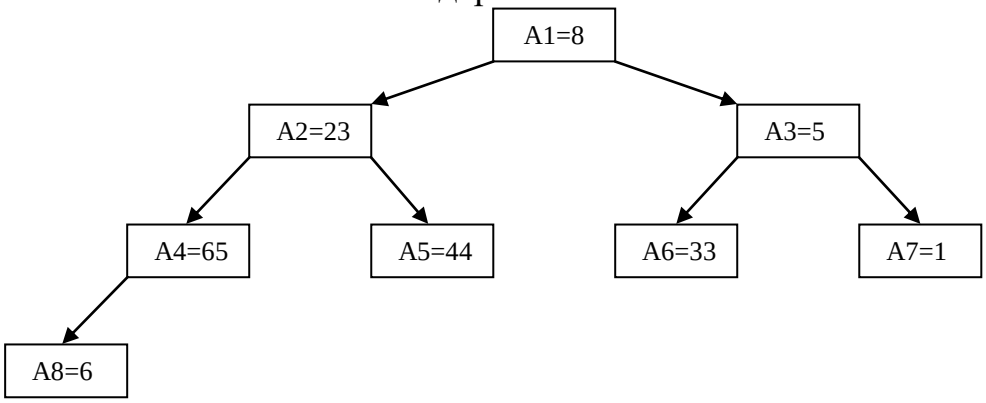


Рис. 2.5. Деревовидне представлення масиву

Очевидно, що при побудові піраміди нас цікавитимуть елементи ***a[n/2], a[n/2–1], ..., a[1]*** для масивів з парним числом елементів і ***елементи a[(n–1)/2], a[(n–1)/2–1], ..., a[1]*** для масивів з непарним числом елементів (оскільки тільки для таких елементів суттєві обмеження піраміди). Припустимо ***i*** – найбільший індекс з числа індексів елементів, для яких суттєві обмеження піраміди. Тоді береться елемент ***a[i]*** в побудованому дереві і для нього виконується процедура просівання, що полягає у тому, що обирається гілка дерева, відповідна ***min(a[2i], a[2i+1])***, та значення ***a[i]*** міняється місцями із значенням відповідного елементу. Якщо цей елемент

не є листом дерева, для нього виконується аналогічна процедура і т.д. Такі дії виконуються послідовно для ***a[i], a[i–1], ..., a[1].*** Легко бачити, що в результаті ми одержимо деревовидне представлення піраміди для початкового масиву.

Приклад сортування за допомогою піраміди надано у таблиці 2.7.

Таблиця 2.7

Приклад сортування за допомогою піраміди

Початкова піраміда	1 6 5 23 44 33 8 65
Крок 1	65 6 5 23 44 33 8 1 5 6 65 23 44 33 8 1 5 6 8 23 44 33 65 1
Крок 2	65 6 8 23 44 33 5 1 6 65 8 23 44 33 5 1 6 23 8 65 44 33 5 1
Крок 3	33 23 8 65 44 6 5 1 8 23 33 65 44 6 5 1
Крок 4	44 23 33 65 8 6 5 1 23 44 33 65 8 6 5 1
Крок 5	65 44 33 23 8 6 5 1 33 44 65 23 8 6 5 1
Крок 6	65 44 33 23 8 6 5 1 44 65 33 23 8 6 5 1
Крок 7	65 44 33 23 8 6 5 1

Алгоритм ***HeapSort*** спочатку будує дерево, що відповідає прямо протилежним співвідношенням: ***A[i] ≥ A[2i], A[i] ≥ A[2i+1],*** а потім змінює місцями ***A[1]*** (найбільший елемент) та ***A[n].***

Як і ***TreeSort***, алгоритм ***HeapSort*** працює в два етапи:

1) етап 1 – побудова сортуючого дерева;

2) етап 2 – просівання елементів по сортуючому дереву.

Дерево, що представляє масив, називається сортуючим, якщо виконуються умови (***A[i] <= A[2i], A[i] < =A[2i+1], A[2i] <= A[2i+1]***). Якщо для деякого ***i*** ця умова не виконується, будемо говорити, що має місце (сімейний) конфлікт у трикутнику ***i***.

Як на І-ом, так і на ІІ-ому етапах елементарна дія алгоритму полягає в вирішенні сімейного конфлікту: якщо найбільший із синів більше, ніж батько, то переставляються батько і цей син (процедура ***ConSwap***).

У результаті перестановки може виникнути новий конфлікт у тому трикутнику, куди переставлений батько. У такий спосіб можна говорити про конфлікт (роду) у піддереві з коренем у вершині ***i***. Конфлікт роду вирішується послідовним вирішенням сімейних конфліктів проходом по дереву униз. Конфлікт роду вирішено, якщо прохід закінчився (***i > n div 2***), або ж в результаті перестановки не виник новий сімейний конфлікт

(процедура ***Conflict***).

```
Procedure ConSwap(i, j : Integer);
  Var b : Real;
  Begin
  If a[i] < a[j] then begin
  b := a[i]; a[i] := a[j]; a[j] := b;
  end;
  End;

Procedure Conflict(i, k : Integer);
  Var j : Integer;
  Begin
  j := 2*i;
  If j = k;
  then ConSwap(i, j)
  else if j < k then begin
   if a[j+1] > a[j] then j := j + 1;
   ConSwap(i, j); Conflict(j, k);
   end;
  End.
```

І етап – побудова сортуючого дерева – оформимо у виді рекурсивної процедури, використовуючи визначення:

Якщо ліве і праве піддерева ***T(2i)*** і ***T(2i+1)*** дерева ***T(i)*** є сортуючими, то для перебудови ***T(i)*** необхідно вирішити конфлікт роду в цьому дереві.

```
Procedure SortTree(i : Integer);
  begin
   If i <= n div 2 then
   begin
    SortTree(2*i);
    SortTree(2*i+1);
    Conflict(i, n)
   end;
  end
```

На 2-му етапі – етапі просівання – для ***k*** від ***n*** до ***2*** повторюються наступні дії:

1.Переставити ***A[1]*** і ***A[k];***

2.Побудувати сортуюче дерево початкового відрізка масиву ***A[1..k–1],*** усунувши конфлікт роду в корені ***A[1].*** Відзначимо, що ***2***-а дія вимагає введення в процедуру Conflict параметра k.

```
   Program HeapSort;
    Const n = 100;
     Var A : Array[1..n] of real;
      k : Integer;
{процедури
ConSwap,Conflict,SortTree,введення,виведення}
     Begin   { введення }
      SortTree(1);
      For k := n downto 2 do begin
      ConSwap(k, 1); Conflict(1, k - 1)
      end;   { виведення }
     End.
```

Приклад програми алгоритму пірамідального сортування:

```
PROGRAM Heapsort (Input,Output);
  const N=8;
  type index=0..N;
    massiv = Array[0..N] of Integer;
   var x:  Integer;
    a:  massiv;
    i,l,r: index;
  PROCEDURE sift; { ---------- }
  label metka;
  var i,j: index;
  BEGIN
  i := l; j := 2*i; x := a[i];
  While j<=r do
   begin
    If j<r then
      If a[j]<a[j+1] then j := j+1;
    If x>=a[j] then Goto metka;
    a[i] := a[j]; i := j; j := 2*i
   end;
  metka: a[i] := x
  END;    { ---------- }

  BEGIN
  For i:=1 to N do ReadLn (a[i]);
```

```
l := (n DIV 2)+1; r := n;
While l>1 do begin l := l-1; sift end;
While r>1 do
begin
 x := a[1]; a[1] := a[r]; a[r] := x;
 r := r-1;
 sift;
end;
For i:=1 to N do Write (a[i],' ')
END.
```

Процедура сортування з використанням піраміди вимагає виконання кроків у кількості порядку (***n log₂ n)*** у найгіршому випадку, це робить її особливо привабливою для сортування великих масивів.

2.14. Сортування із злиттям

Сортування із злиттям, як правило, застосовуються в тих випадках, коли вимагається відсортувати послідовний файл, що не вміщується цілком в основній пам'яті (у методах зовнішнього сортування). Проте існують і ефективні методи внутрішнього сортування, засновані на розбитті і злитті.

Поширеним алгоритмом внутрішнього сортування (для простоти вважатимемо, що число елементів в масиві, є ступенем числа ***2***) є: два відсортованих у порядку зростання масиви ***p[1], p[2], ..., p[n]*** та ***q[1], q[2], ..., q[n]*** і є порожній масив ***r[1], r[2], ..., r[2n],*** який ми хочемо заповнити значеннями масивів ***p*** та ***q*** у порядку зростання. Для злиття порівнюються ***p[1]*** та ***q[1],*** менше із значень записується в ***r[1]***; якщо це значення ***p[1]***, тоді ***p[2]*** порівнюється з ***q[1]*** і менше із значень заноситься в ***r[2]***; якщо це значення ***q[1]***, тоді на наступному кроці порівнюються значення ***p[2]*** та ***q[2]*** і т.д., доки ми не досягнемо меж одного із масивів. Тоді залишок іншого масиву просто дописується в ***«хвіст»*** масиву ***r***. Приклад злиття двох масивів наведено на рис. 2.16.

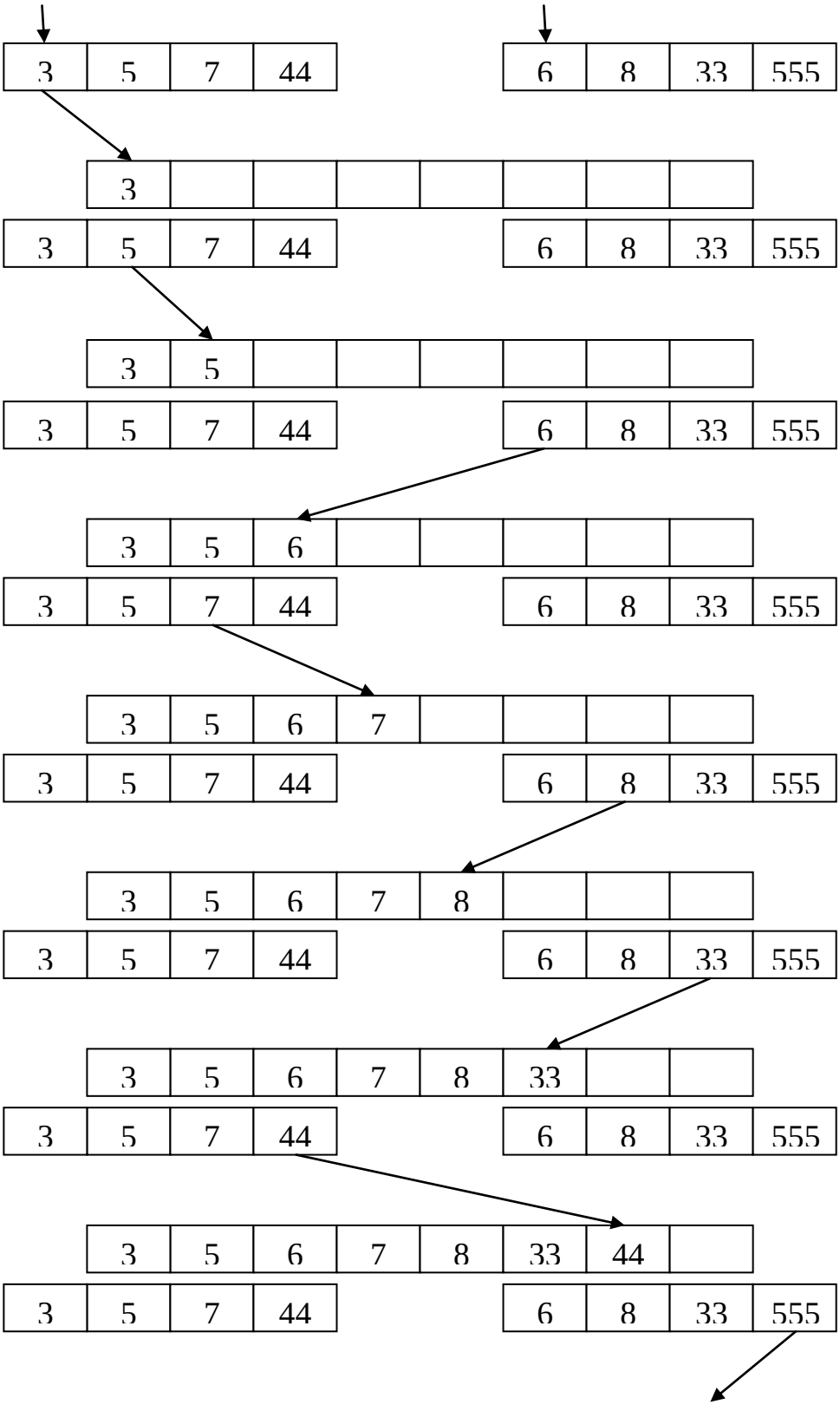
Для сортуванням із злиттям масиву ***a[1], a[2], ..., a[n]*** відкривається додатковий масив ***b[1], b[2], ..., b[n]*** та виконуються наступні кроки:

1) злиття ***a[1]*** та ***a[n]*** з розміщенням результату в ***b[1], b[2]***, злиття ***a[2]*** та ***a[n–1]*** із розміщенням результату в ***b[3], b[4],*** ..., злиття ***a[n/2]*** та ***a[n/2+1]*** з розміщенням результату в ***b[n–1], b[n];***

2) злиття пар ***b[1], b[2]*** та ***b[n–1], b[n]*** з розміщенням результату в ***a[1], a[2], a[3], a[4],*** злиття пар ***b[3], b[4]*** та ***b[n–3], b[n–2]*** з розміщенням результату в ***a[5], a[6], a[7], a[8], ...,***

3) злиття пар ***b[n/2–1], b[n/2]*** та ***b[n/2+1], b[n/2+2]*** з розміщенням результату в ***a[n–3], a[n–2], a[n–1], a[n],*** і т.д.

4) на останньому кроці, наприклад, (залежно від значення ***n***), виконується злиття послідовностей елементів масиву ***завдовжки n/2 a[1], a[2], ..., a[n/2]*** та ***a[n/2+1], a[n/2+2], ..., a[n]*** з розміщенням результату в ***b[1], b[2], ..., b[n].***



3	5	6	7	8	33	44	555

Рис. 2.6. Приклад злиття двох масивів

При застосуванні сортування із злиттям число порівнянь ключів і число пересилок оцінюється як ***O(n log n).*** Проте слід враховувати, що для виконання алгоритму для сортування масиву розміру ***n*** потрібно ***2n*** елементів пам'яті.

Приклад сортування із злиттям наведено у таблиці 2.8.

Таблиця 2.8

Приклад сортування із злиттям

Початковий стан масиву	8 23 5 65 44 33 1 6
Крок 1	6 8 1 23 5 33 44 65
Крок 2	6 8 44 65 1 5 23 33
Крок 3	1 5 6 8 23 33 44 65

Процедура Mergesort виконує сортування двох масивів «злиттям». Тобто, спочатку перший масив сортується за допомогою процедури ***Qsort*** (наприклад, швидке сортування) і другий масив сортується за допомогою цієї ж процедури. Потім два, вже відсортованих масиви з'єднуються в один, а саме, за допомогою оператору ***«if»*** ми порівнюємо перший елемент одного і перший елемент другого масивів. Якщо один з них менший, то ми відсилаємо його в третій масив, а індикатор пересуваємо на наступний елемент і знов порівнюємо їх. Знову менший з двох елементів пересилаємо в третій масив, а індикатор пересуваємо. Також ми передбачили варіант, коли елементи, що порівнюються – однакові. Тоді в третій масив по–черзі записуються обидва елементи, а індикатори зміщуються на наступні елементи в обох масивах. Якщо один з масивів виявився меншим за кількістю елементів ніж інший, то решта елементів довшого масиву просто пересилається до третього масиву в тій самій послідовності (адже вхідні масиви вже відсортовані раніше процедурою ***Qsort***). Таким чином, в результаті ми отримуємо третій масив з впорядкованими елементами.

```
Procedure Mergesort;
    Var i, j, k: byte;
Begin
    i:=1;
    j:=1;
    for k:=1 to n_c do c[k]:=0;
    k:=1;
```

```
While ( i<= n_a ) or ( j<=n_b ) do
if i= n_a+1 then

begin
 c[k]:=b[j];
 k:=k+1;
 j:= j+1;
 end;
else
if i= n_b+1 then
 begin
  c[k]:=a[i];
  k:= k+1;
  i:=i+1 ;
 end;
else
 if a[i]= b[j] then
  begin
   c[k]:=b[j];
   c[k+1]:= a[i];
   k:=k+2;
   i:=i+1;
   j:=j+1
   end;
  else
    if a[i] > b[j] then
    begin
     c[k]:= b[j];
     k:= k+1;
     j:= j+1;
    end;
    else
    begin
     c[k]:= a[i];
     k:= k+1;
     i:= i+1;
    end;
  End;
```

2.15. Порозрядне цифрове сортування

Алгоритм порозрядного цифрового сортування істотно відрізняється від описаних раніше.

По-перше, він зовсім не використовує порівнянь елементів, які сортуються.

По-друге, ключ, по якому відбувається сортування, необхідно розділити на частини, розряди ключа. Наприклад, слово можна розділити по буквах, число – по цифрах... До сортування необхідно знати два параметри: ***k*** та ***m***, де ***k*** – кількість розрядів у найдовшому ключі; ***m*** – розрядність даних – кількість можливих значень розряду ключа. Наприклад, при сортуванні кириличних слів ***m = 33***, оскільки буква може приймати не більш ***33*** значень. Якщо в найдовшому слові ***10*** букв, ***k = 10***. Аналогічно, для шістнадцяткових чисел ***m= 16***, якщо як розряд брати цифру, та ***m=256***, якщо використовувати побайтове розподілення.

По-третє, параметри ***k*** та ***m*** не можна змінювати в процесі роботи алгоритму.

Група методів порозрядного цифрового сортування використовує стратегію, прямопротилежну методам злиття. Ключі ***A[i]*** елементів файлу розглядаються в цьому алгоритмі як деякі послідовності із ***s*** символів у відомому алфавіті ***K=(A1,A2,A3,...,As).*** Наприклад, такий порядок використовується для розташування слів у словнику. Якщо множину слів упорядкувати спочатку по правій букві, потім, не змінюючи порядку, упорядкувати їх по наступній букві і т.д. до кінця слова, то слова будуть розташовані в лексикографічному порядку. Можна починати вибір букв як з кінця, так і з початку слів. В процесі впорядковування по букві множина слів розбивається на підмножини з однаковими значеннями даної букви, а потім підмножини знову об’єднуються в одну множину в порядку, відповідному алфавітному порядку для значень цієї букви. Розподіл множини по підмножинах і дало назву цієї групи методів – ***розподілююче сортування (RadixSort).***

Приклад розподілюючого сортування:

Область пам’яті, яку займає масив перерозподіляється між вхідною і вихідною множинами. Вихідна множина (вона розміщується на початку масиву) розбивається на групи. Розбиття відстежується в масиві ***b***. Елемент масиву ***b[i]*** містить індекс в масиві ***a***, з якого починається ***(i+1)***-а група. Номер групи визначається значенням проаналізованої цифри числа, тому індексація в масиві ***b*** починається з ***0***. Коли чергове число обирається із вхідної множини ***i*** та повинно бути занесене в ***i***-у групу вихідної множини, воно буде записане в позицію, яка визначається значенням ***b[i]***. Проте заздалегідь ця позиція повинна бути звільнена: ділянка масиву від ***b[i]*** до кінця вихідної множини включно зсовується праворуч (у бік молодших розрядів). Після запису числа в ***i***-у групу ***i***-е та усі подальші значення в масиві ***b*** корегуються – збільшуються на ***1.***

```
Program RadixSort;
    {Цифрове сортування (розподіленням)}
     const D=...; {максимальна кількість цифр у числі}
   P=...;   {основа системи числення}
Procedure RadixSort(var a : Seq);
  Var b : array[0..P-2] of integer;
{індекс елементу,що слідує за останнім в і-й групі}
  i, j, k, m, x ,v : integer;
begin
  for m:=1 to D do begin
    {перебирання цифр, починаючи з молодшої}
  for i:=0 to P-2 do b[i]:=1;
       {початк. значення індексів}
  for i:=1 to N do begin  {перебирання масиву}
        {визначення m-ї цифри}
  v:=a[i];
  for m:=m downto 2 do v:=v div P;
  k:=v mod P;
  x:=a[i];
   {зсув-вивільнення місця в кінці k-оїї групи}
  for j:=i downto b[k]+1 do a[j]:=a[j-1];

                              {запис в кінець k-оїї групи}
  a[b[k]]:=x;
                  {модифікація k-го індексу та більших}
  for j:=k to P-2 do b[j]:=b[j]+1;
  end;
 end;
```

У таблиці 2.9 надано результати прикладу розподілюючого сортування при *P=10* та *D=4*:

Таблиця 2.9

Результати прикладу розподілюючого сортування при P=10 та D=4

Цифра	Зміст масивів ***a*** та ***b***
Вхідний масив	220 8390 9524 9510 462 2124 7970 4572 4418 1283

1	220 8390 9510 7970 462 4572 1283 9524 2124 4418 +———0———+ +—2—+ +–3+ +——4—+ +–8+ b=(5,5,7,8,10,10,10,10,11,11)
2	9510 4418 220 9524 2124 462 7970 4572 1283 8390 +—1—+ +———2———+ +–6+ +——7—+ +–8–+ +–9+ b=(1,3,6,6,6,6,7,9,10,11)
3	2124 220 1283 8390 4418 462 9510 9524 4572 7970 +–1+ +—2——+ +–3+ +——4—+ +———5———+ +–9+ b=(1,2,4,5,7,10,10,10,10,11)
4	220 462 1283 2124 4418 4572 7970 8390 9510 9524 +—0—+ +–1+ +–2+ +——4——+ +–7+ +–8+ +—9——+ b=(3,4,5,5,7,7,7,8,9,11)

Приклад реалізації алгоритму «розподілюючого» сортування з використанням підпрограми:

```
Prrogram RadixSortPP;

Const n = 8;
Type arrType = Array[0 .. Pred(n)] Of Byte;
Const m = 256;
  a: arrType = (44, 55, 12, 42, 94, 18, 6, 67);

Procedure RadixSort(Var source, sorted: arrType);
Type indexType = Array[0 .. Pred(m)] Of Byte;
Var distr, index: indexType;
     i: integer;

begin
 fillchar(distr, sizeof(distr), 0);
 for i := 0 to Pred(n) do inc(distr[source[i]]);
 index[0] := 0;
 for i := 1 to Pred(m) do
 index[i] := index[Pred(i)] + distr[Pred(i)];
 for i := 0 to Pred(n) do begin
 sorted[ index[source[i]] ] := source[i];
 index[source[i]] := index[source[i]]+1;
 end;
end;

var b: arrType;
begin
  RadixSort(a, b);
```

```
end.
```

Час роботи алгоритму приблизно оцінюється як: ***(kn+s),*** де ***k, s*** – константи, що залежать від програмної реалізації алгоритму.

2.16. Сортування підрахунком

Сортування підрахунком (***CountSort)*** – спеціалізований алгоритм, який працює неймовірно швидко, якщо елементами масиву є цілі числа, із значеннями, які займають, відносно вузький діапазон (діапазон значень повинен бути порівнянний з довжиною масиву). Доки виконуються ці умови алгоритм працює відмінно. Для прикладу можна привести результат сортування 1-го мільйона елементів із значенням від ***1–10000*** (комп'ютер з процесором Pentium–133). Час швидкого сортування був рівний ***3,93*** секунди, результат же сортування підрахунком був ***0,37*** секунди, тобто швидше в ***10*** разів [40].

Велика швидкість виконання досягає за рахунок того, що в алгоритмі не використовуються операції порівняння. Без них алгоритм сортування може упорядковувати значення за час рівний ***O(n).***

Приклад процедури сортування підрахунком:

```
procedure CountSort(var ar: array of integer;
     min, max: integer);
var  counter: array[0..100000] of integer;
  i, j, index: Integer;

begin
  // заповнюємо масив нулями
  for i:=0 to high(counter) do tmpA[i ]:=0;
  for i:=min to max do
  begin
   counter[ar[ i]]:=counter[ar[i ]]+1;
  end;
  // встановлюємо значення в правильну позицію
  index:=min;
  for i:=min to high(counter)-1 do
  begin
   for j:=0 to counter[ i]-1 do
   begin
     ar[index]:=i;
```

```
      index:=index+1;
    end;
  end;
end;
```

Спочатку створюється масив для підрахунку елементів, які мають певне значення, і встановлюємо всі значення рівними нулю. Потім алгоритм обчислює скільки разів у списку зустрічається кожен елемент і збільшує значення відповідного запису лічильника на 1. Іншими словами, при дослідженні елементу масиву під номером ***i*** програма збільшує значення ***counter[ar[i]].*** Алгоритм проходить через весь масив лічильників, уміщуючи певне число елементів у відсортований масив. Для кожного значення ***i*** від ***1*** до ***N*** він додає ***counter[i]*** елементів із значенням ***i***.

2.17. Оболонкове сортування

Оболонкове сортування іноді називають сортуванням включеннями з убуваючим приростом [42]. Метод сортування вставкою працює поволі тому, що він обмінює тільки суміжні елементи. Оболонкове сортування є простим розширенням сортування вставкою яка збільшує швидкість роботи алгоритму за рахунок того, що дозволяє обмінювати елементи, які знаходяться далеко один від одного.

Основною ідеєю алгоритму є відсортувати всі групи файлу що складаються з елементів файлу віддалених один від одного на відстань ***h.*** Такі файли називаються ***h***-відсортованими. Коли виконується ***h***-сортування файлу по деякому великому ***h***, відбувається пересування елементів файлу на великі відстані. Легше робити сортування для менших значень ***h***. Процес закінчується коли ***h*** зменшується до ***1***.

Приклад оболонкового сортування:

```
31 59 54 39 22 83 80 2 6 60
22 59 54 39 31 83 80 2 6 60
22 59 54 2 31 83 80 39 6 60
22 59 54 2 6 83 80 39 31 60
6 59 54 2 22 83 80 39 31 60
6 59 54 2 22 60 80 39 31 83
6 54 59 2 22 60 80 39 31 83
6 54 2 59 22 60 80 39 31 83
6 2 54 59 22 60 80 39 31 83
```

```
    2 6 54 59 22 60 80 39 31 83
    2 6 54 22 59 60 80 39 31 83
    2 6 22 54 59 60 80 39 31 83
    2 6 22 54 59 60 39 80 31 83
    2 6 22 54 59 39 60 80 31 83
    2 6 22 54 39 59 60 80 31 83
    2 6 22 39 54 59 60 80 31 83
    2 6 22 39 54 59 60 31 80 83
    2 6 22 39 54 59 31 60 80 83
    2 6 22 39 54 31 59 60 80 83
    2 6 22 39 31 54 59 60 80 83
    2 6 22 31 39 54 59 60 80 83
    2 6 22 31 39 54 59 60 80 83
```

Приклад процедури оболонкового сортування:

```
procedure CoverSort;
var i,j,z,k:word;
 v:integer;
begin
 z:=1;
 k:=3;
 repeat
  z:=z*k+1;
 until z>n;
 repeat
  z:=z div k;
  for i:=z+1 to n do
  begin
   j:=i;
   while (j>z) and (a[j-z]>a[j]) do
   begin
    v:=a[j];
    a[j]:=a[j-z];
    j:=j-z;
    a[j]:=v;
   end;
  end;
 Until z=1;
end;
```

У прикладі показано, як оболонкове сортування обробляє файл з

кроком в ..., ***1093, 364, 121, 40, 13, 4, 1***. Можуть існувати і інші послідовності – одні кращі, інші гірші. *Властивість* оболонкового сортування полягає у тому, що воно ніколи не робить більш ніж ***N1.5*** порівнянь для вищезгаданої послідовності ***h***.

2.18. Сортування підрахунком розподілень

Якщо відомо інтервал зміни значень в послідовності і значення часто повторюються, то застосовують алгоритм сортування підрахунком розподілень [42] .

Приклад процедури сортування підрахунком розподілень:

```
procedure CountDistribSort;{a:[7 8 2 9 3 1 5 8 3 3]}
var
  i,j: integer;
begin
 for j:=0 to M-1 do
  count[j]:=0;
 for i:=1 to N do
  inc(count[a[i]]);{count:[0 1 1 3 0 1 0 1 2 1 0]}
 for j:=0 to M-1 do
  count[j]:=count[j-1]+count[j];
     {count:[0 1 2 5 5 6 6 7 9 10 10]}
 for i:=N downto 1 do
  begin
   b[count[a[i]]]:= a[i];
   dec(count[a[i]]);
  end;
 a := b; {a:[1 2 3 3 3 5 7 8 8 9]}
end;
```

2.19. Порівняння методів сортування

Прийнято метод сортування характеризувати по чотирьох параметрах:

1. ***Час*** сортування.
2. ***Пам'ять*** – додаткові витрати пам'яті, які залежать від розміру масиву.
3. ***Стійкість*** – стійке сортування не міняє взаємного розташування рівних елементів. Багато методів (наприклад, ***HeapSort***) по ходу роботи так перемішують масив, що рівні елементи після сортування

можуть йти в зовсім іншій послідовності, ніж до цього.

4. ***Природність*** поведінки – ефективність методу при обробці вже відсортованих, або частково відсортованих даних. Природній метод враховує цей факт і працює швидше. Неприродний метод цей факт не враховує, або буває, що працює навіть *гірше*.

Крім того, виділяють два основних типи сортувань (табл. 2.10).

Таблиця 2.10

Основні типи сортувань

Розподілююче та його варіації	Сортування порівняннями
Заснований на тому, що число можливих значень ключа – кінцеве число.	Використовується лише можливістю прямого порівняння ключів та їх впорядкованістю. ***Quicksort, Heapsort, Insertsort ...***

Методи сортування ***SelectSort, BubbleSort, ShellSort*** практично не використовуються.

Сортування двійковим деревом ***TreeSort***

1. Загальна швидкодія завжди ***O(nlogn).***
2. Звичайне дерево: ***n*** елементів (***ключ + 2*** покажчики), вибір із заміщенням: ***2n–1*** елементів.
3. Стійкості немає.
4. Поведінка неприродня.

Застосовують там, де:

1) побудоване дерево можна з успіхом застосувати для інших задач;
2) дані вже побудовані у *‘дерево’*;
3) дані можна прочитувати безпосередньо у дерево.

Наприклад, при потоковому введенні з консолі або з файлу.

InsertSort – прості вставки.

1. Загальна швидкодія – $O(n^2)$, але зважаючи на простоту реалізації методу є найшвидшим для малих (***12–20*** **елементів**) масивів та списків.
2. Сортування відбувається *‘на місці’* – додаткових витрат пам’яті немає.
3. Стійка.
4. Природня поведінка.

Метод простої вставки ***InsertSort*** добре використовувати для частково відсортованих даних.

QuickSort – швидке сортування

1. Загальна швидкодія ***Quicksort – O(nlogn).*** Випадок $O(n^2)$ можливий лише теоретично, але вірогідність такого випадку надзвичайно

мала. Загалом – найшвидше сортування порівняннями для невпорядкованих масивів із ***50*** та більше елементів.

2. Ітераційний варіант вимагає ***(log n)*** пам'яті, рекурсивний – ***n.***

3. Стійкості немає.

4. Поведінка неприродна. Вже практично відсортований масив сортуватиметься стільки ж, скільки і повністю невпорядкований.

HeapSort – пірамідальне сортування.

1. У 1.5 рази повільніше швидкого сортування. Загальна швидкодія завжди ***O(nlogn).***

2. Сортування 'на місці'

3. Стійкості немає.

4. Поведінка неприродня.

Основна перевага – сортування не вимагає додаткової пам'яті, хороший середній випадок. Його елементи часто застосовуються в суміжних задачах.

MergeSort – сортування злиттям.

1. Загальна швидкодія – ***O(nlogn).***

2. Вимагає ***(n)*** пам'яті.

3. Стійка.

4. Залежить від конкретного сортування, заснованого на принципі злиття.

На практиці використовуються реалізації з природньою поведінкою. ***MergeSort*** – загальна назва для обширного класу сортувань, заснованих на принципі злиття. Такі сортування поступаються ***QuickSort*** при роботі з масивами, але виграють на файлах і списках.

RadixSort – Розподілююче сортування.

1) Для масивів: є всього ***n*** елементів, сортування виконується по ***k*** розрядів, кожен може приймати до ***s*** різних значень; числа ***k*** та ***s*** – константи, відомі до початку сортування.

1. Загальна швидкодія ***O(k(n+s))*** [***s*** іноді не враховують зовсім]

2. Реалізація з тимчасовим масивом ***O(n)***, [***RadixSort***],
реалізація без тимчасового масиву ***O(logn)*** [***RadixExchangeSort***]

3. Реалізація з тимчасовим масивом стійка,
реалізація без тимчасового масиву нестійка.

4. Неприродна поведінка.

2) Для списків.

1. Загальна швидкодія ***O(k(n+s))*** [s іноді не враховують].

2. Додаткової пам'яті не вимагається, оскільки

реорганізовувати список можна просто через покажчики.

3. Стійке сортування.

4. Неприродна поведінка.

Який метод сортування найшвидший?

У загальному випадку відповідь наступна:

1) для малих масивів та списків (менше 20 елементів) – ***InsertSort***;

2) для великих масивів (більше 20 елементів) – ***QuickSort***;

3) для довгих списків (більше 20 елементів) та файлів – ***MergeSort***.

Для сортувань порівняннями давно доведена теорема про максимальну швидкодію: ***Omega(nlogn)*** операцій.

Якщо кількість можливих різних ключів ненабагато більше їх загального числа – то найшвидшою буде розподілююче сортування та його варіації (***RadixSort*** – ‘радіксна’). Наприклад, для великих масивів рядків, цілих чисел з невеликого проміжку. На принципах ***RadixSort*** засновано метод ***MultiwayQuicksort /MQS/*** – швидке сортування із складовими ключами (автори Jon Bentley і Robert Sedgewick). ***MQS*** поєднав переваги ***RadixSort*** та ***QuickSort***.

Існує ще дуже багато різних методів сортувань, які виходять за рамки розглянутих вище. У кожному конкретному випадку треба дивитися на тип та розмір даних та на наявні ресурси. У типових випадках найпростішими і достатньо швидкими рішеннями будуть стандартні методи.

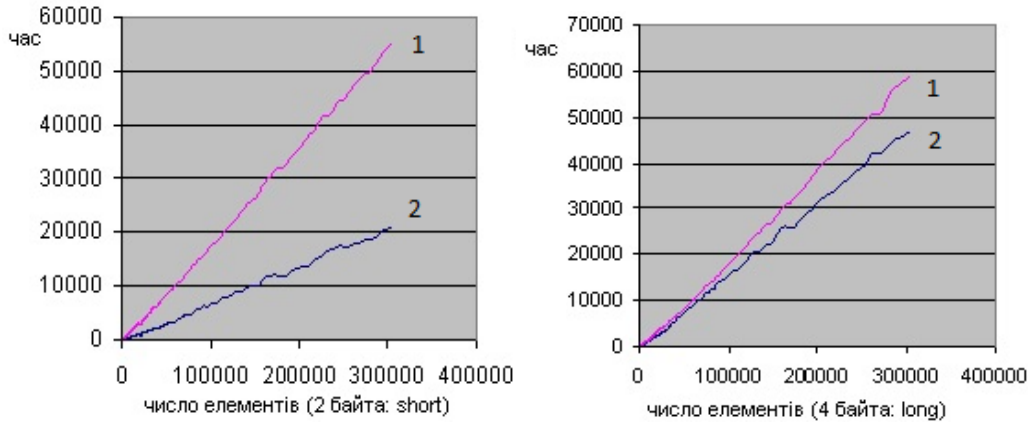
Ефективність порозрядного сортування

Час виконання порозрядного сортування росте лінійним чином по ***n***, оскільки ***k, m*** – константи [1-6]. Також час виконання зростає лінійним чином по k – при збільшенні довжини типу (кількості розрядів) відповідно зростає число проходів, але з деякими корективами, що залежать від реальних умов. Річ у тому, що основний цикл сортування по ***i***–му байту полягає в русі покажчика кроками ***sizeof(T)*** по масиву для отримання чергового розряду. Коли ***T=char***, крок дорівнює ***1***, ***T=short*** – крок дорівнює ***2***, ***T=long*** – крок дорівнює ***4***... Чим більше розмір типу, тим менш послідовний доступ до пам’яті здійснюється, а це вельми впливає на швидкість роботи. Тому реальний час зростає трохи швидше, ніж його теоретичне значення.

Крім того, порозрядне сортування вже за своєю суттю *нецікаве* для малих масивів. Кожен прохід містить мінімум ***256*** ітерацій, тому при невеликих ***n*** загальний час виконання ***(n+m)*** визначається константою ***m=256***.

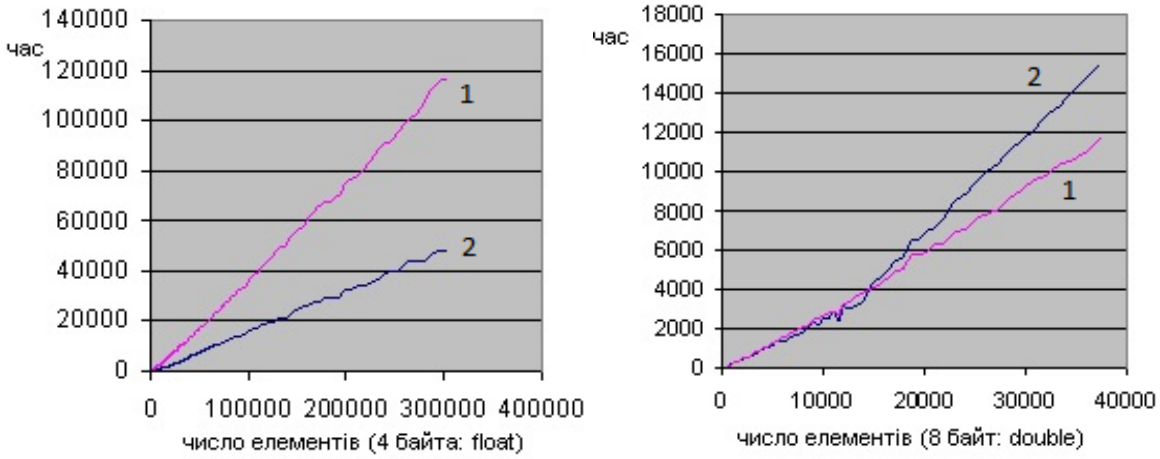
На рис. 2.7 та рис. 2.8 зображені результати порівнянь швидкого сортування (лінія 1) і порозрядного сортування (лінія 2) для різних типів

даних.



а) б)

Рис. 2.7. Результати порівнянь швидкого сортування (лінія 1) і порозрядного сортування (лінія 2):
а) тип даних short; б) тип даних long.



а) б)

Рис. 2.8. Результати порівнянь швидкого сортування (лінія 1) і порозрядного сортування (лінія 2):
а) тип даних float; б) тип даних double.

Short **–** Завдяки малій розрядності і простоті внутрішніх циклів, порозрядне сортування випередило швидке сортування вже на 100 елементах.

Long **–** При переході до типу ***Long*** відбулося двократне збільшення кількості проходів ***RadixPass***, тому гірша асимптотика швидкого сортування показала себе лише після 600 елементів, де вона почала відставати.

Float **–** По графіку видно, що для цього типу порозрядне сортування виявилося набагато ефективніше, ніж для ***Long***. Справа у тому, що на типі ***Float*** різко зростає час роботи швидкого сортування (хоча розмір

однаковий – 4 байти). Це пов’язано з тим, що процесор, перш ніж робити які-небудь операції, переводить його в ***Double***, а потім працює з цим типом, який у 2 рази більший за розміром.

Так чи інакше, швидке сортування стало працювати у 2 рази повільніше, а порозрядне сортування вийшло уперед після ***100*** елементів.

Double – Графік показаний з десятиразовим збільшенням. Розмір типу збільшився в 2 рази, тому для невеликих ***n*** порозрядне сортування, звичайно, відстає, але випередить швидке сортування на масивах з декількох тисяч елементів.

Невелика перевага ***RadixSort*** триває приблизно до ***16000*** елементів, коли дані починають виходити за межі кеша. Відповідно, починає відігравати свою роль повільний непослідовний доступ (з кроками по 8 байт) до основної пам’яті. В результаті порозрядне сортування знову вибивається уперед лише після ***500000*** елементів.

Що краще: розподілююче сортування чи сортування порівняннями?

Коли кількість можливих різних ключів ненабагато більша за їх загальне число – ***розподілююче сортування***.

Коли кількість можливих різних ключів дуже велика, а розмір масиву порівняльно невеликий – ***сортування порівняннями***.

Приклад співвідношення часу сортування випадкових цілих чисел з проміжку [0..999] різними методами сортування:

n	10	20	30	40	50	100	200
QuickSort	0.22	0.55	0.88	1.65	2.14	4.50	13.13
MergeSort	0.33	0.88	1.37	1.97	2.74	6.37	14.78
RadixSort	0.38	0.71	0.99	1.38	1.70	3.29	6.64

Цифри можуть змінюватися залежно від реалізації (потужність комп’ютера, варіант компілятора і т.ін.), але співвідношення залишаться такими ж.

Приклад-ілюстрація уповільнення розподілюючого сортування при розширенні проміжку у два рази (метод працює у 2 рази повільніше)

	проміжок 0–999			проміжок 0–999999		
	Порів–нянь	Перемі–щень	Час	Порів–нянь	Перемі–щень	Час
QuickSort	600	478	4.51	846	562	5.93
MergeSort	543	672	6.43	538	672	6.38
RadixSort	0	1000	3.29	0	2000	6.92

Для простих методів сортування існують точні формули, за якими обчислюються мінімальне, максимальне і середнє число порівнянь ключів ***(C)*** та пересилань елементів масиву ***(M)*** – табл. 2.11.

Таблиця 2.11

Характеристики простих методів сортування

		Min	Avg	Max
Пряме включення	C M	n–1 2(n–1)	$(n^2 + n – 2)/4$ $(n^2 – 9n – 10)/4$	$(n^2 –n)/2 – 1$ $(n^2 –3n – 4)/2$
Прямий вибір	C M	$(n^2 – n)/2$ 3(n–1)	$(n^2 – n)/2$ n(ln n + 0.57)	$(n^2 – n)/2$ $n^2/^4 + 3^x(n–1)$
Прямий обмін	C M	$n^2 – n)/2$ 0	$(n^2 – n)/2$ $(n^2 – n)0.75$	$(n^2 – n)/2$ $(n^2 – n)1.5$

Для оцінок складності вдосконалених методів сортування точних формул немає. Відомо лише, що для сортування методом Шелла порядок ***C*** та ***M*** є ***O(n(1.2))*** чи ***O(n(1.25),*** а для методів ***QuickSort***, ***HeapSort*** і сортування із злиттям ***MergeSort*** – ***O(nlogn).*** Проте результати експериментів свідчать про те, що ***QuickSort*** показує результати в ***2-3*** рази кращі, ніж ***HeapSort*** (табл. 2.12 містить вибірку результатів, які одержані при прогоні програм, написаних на мові ***Модула-2***). Саме ***QuickSort*** звичайно використовується в стандартних утилітах сортування (зокрема, в утиліті ***sort***, що поставляється з операційною системою ***UNIX***).

Для проведення експериментального порівняльного аналізу різних методів сортування була розроблена програма, яка в автоматичному режимі підраховувала необхідний для кожного методу сортування час. Для чистоти експерименту сортування усіма методами проводилося на однакових наборах вхідних даних, потім формувався новий набір даних і вони знову піддавалися сортуванню різними методами. Таким чином, було виконано ***60*** циклів сортування, підраховано середній час, який потрібно кожному методу, щоб відсортувати вхідний масив. Для повнішого аналізу методів в кожному циклі сортування проводилося для розмірностей ***500, 1000, 3000, 5000, 8000, 10000, 30000, 60000***. Це дало можливість прослідкувати динаміку зростання необхідного для сортування часу. Також перевірялося, як поводяться методи на різних вхідних даних: 1) впорядкованих у випадковому порядку (невпорядковані); 2) впорядкованих у прямому порядку; 3) впорядкованих у зворотному порядку. У таблицях 2.12-2.14 наведені результати експериментального підрахунку часу сортування масиву, заповненого елементами у випадковому порядку (невпорядковані), впорядкованих у прямому порядку та впорядкованих у зворотному порядку відповідно. У табл. 2.12

наведено теоретичну складність основних методів сортування.

Таблиця 2.12

Час роботи програм сортування

	Впорядкований масив	Невпорядкований масив	Масив впорядковано у зворотньому порядку
	n = 256		
Heapsort	0.20	0.08	0.18
Quicksort	0.20	0.12	0.18
Сортування злиттям	0.20	0.08	0.18
	n = 2048		
Heapsort	2.32	0.72	1.98
Quicksort	2.22	1.22	2.06
Сортування злиттям	2.12	0.76	1.98

Таблиця 2.13

Результати для масиву, заповненого випадковими елементами

Метод сортування	Розмірність масиву сортування							
	500	1000	3000	5000	8000	10000	30000	60000
Підрахунком	0.08	0.31	2.76	7.67	19.67			
Включенням	0.03	0.06	0.60	1.66	4.22	6.58	59.62	
Шелла	0.02	0.05	0.43	1.12	2.72	4.24	35.62	
Злиттям	0.03	0.02	0.06	0.09	0.11	0.13	0.38	0.77
Екстракцією	0.01	0.10	0.92	2.54	6.51	10.17		
Деревом	0.02	0.02	0.11	0.20	0.33	0.42	1.39	3.02
Бульбашковий	0.07	0.27	2.43	6.75	17.28			
Швидке	0.00	0.00	0.01	0.03	0.03	0.04	0.13	0.27
Двійкове розподілення	0.00	0.00	0.02	0.03	0.05	0.06	0.20	0.38
Вироджений розподіл	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02

Таблиця 2.14

Результати для масиву, заповненого початково впорядкованими елементами

Метод сортування	Розмірність масиву сортування							
	500	1000	3000	5000	8000	10000	30000	60000
Підрахунком	0.08	0.32	2.65	7.26	18.46			
Включенням	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03
Шелла	0.00	0.00	0.01	0.02	0.02	0.05	0.15	0.29
Злиттям	0.02	0.02	0.04	0.08	0.09	0.12	0.31	0.63
Екстракцією	0.02	0.11	0.92	2.54	6.53	10.20		
Деревом	0.01	0.03	0.11	0.19	0.34	0.43	1.45	3.11
Бульбашковий	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02
Швидке	0.00	0.00	0.01	0.02	0.03	0.04	0.11	0.22
Двійкове розподілення	0.00	0.01	0.02	0.03	0.04	0.05	0.18	0.36

Вироджений розподіл	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02

Таблиця 2.15

Результати для масиву, заповненого початково впорядкованими в зворотному порядку елементами

Метод сортування	Розмірність масиву сортування							
	500	**1000**	**3000**	**5000**	**8000**	**10000**	**30000**	**60000**
Підрахунком	**0.09**	**0.32**	**2.64**	**7.26**	**18.47**			
Включенням	**0.02**	**0.13**	**1.16**	**3.26**	**8.37**	**13.08**		
Шелла	**0.01**	**0.00**	**0.03**	**0.16**	**0.21**	**0.27**	**2.09**	**8.02**
Злиттям	**0.02**	**0.03**	**0.06**	**0.06**	**0.10**	**0.11**	**0.32**	**0.64**
Екстракцією	**0.02**	**0.11**	**0.91**	**2.54**	**6.51**	**10.19**		
Деревом	**0.00**	**0.02**	**0.10**	**0.18**	**0.30**	**0.39**	**1.32**	**2.85**
Бульбашковий	**0.06**	**0.29**	**2.95**	**8.35**	**21.60**			
Швидке	**0.00**	**0.00**	**0.01**	**0.01**	**0.03**	**0.04**	**0.11**	**0.24**
Двійкове розподілення	**0.01**	**0.01**	**0.03**	**0.03**	**0.05**	**0.07**	**0.19**	**0.37**
Вироджений розподіл	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.01**	**0.03**

На рис. 2.9 наведено приклад співвідношення у часі деяких алгоритмів сортування (1 - бульбашкове сортування; 2 - шейкер-сортування; 3 - сортування вибором; 4 - сортування вставками; 5 - сортування вставками з сторожевим елементом; 6 - сортування Шелла)

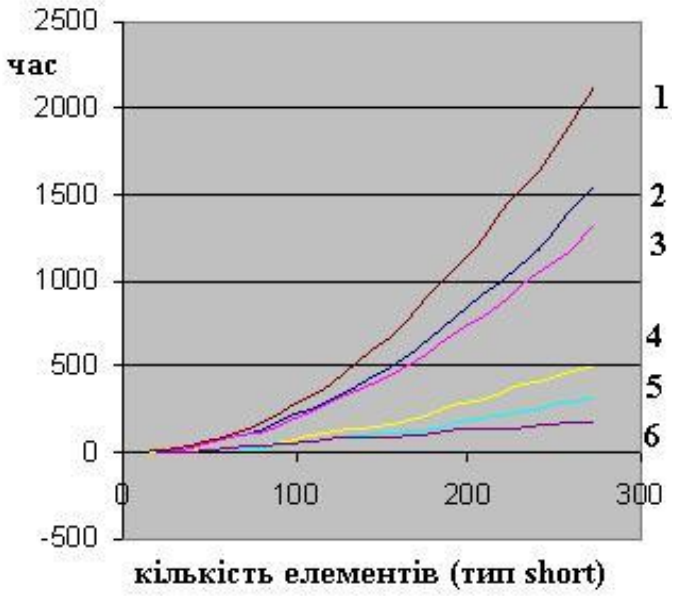


Рис. 2.9. Приклад співвідношення у часі деяких алгоритмів сортування:
1 - бульбашкове сортування;
2 - шейкер-сортування;
3 - сортування вибором;
4 - сортування вставками;
5 - сортування вставками з сторожевим елементом;
6 - сортування Шелла.

Для порожніх комірок випробування не проводилося. Час засікався з точністю до 1/50 с (нулі означають, що було витрачено менше часу, ніж 1/50 с). Масиви розміщувалися в динамічній пам'яті.

Особливості, що виникають при сортуванні, узагальнені у табл. 2.16.

Таблиця 2.16

Теоретична складність деяких основних сортування

Метод сортування	Теоретична складність			Пам'ять
	T_{max}	T_{mid}	T_{min}	O_{max}
1	2	3	4	5
Підрахунком /п*одсчетом/ CountSort*	n^2			n
Бульбашковий (простим вибором, лінійне сортування) /*пузырьковая сортировка/ BubbleSort*	n^2		n	1
Бульбашковий з просіванням */пузырьковая с просеиванием/ BubbleSiftingSort*	$4/7 \cdot n^2$		$4/7 \cdot n$	1

Продовження табл. 2.16

1	2	3	4	5
Шейкер /*шейкер/ ShakerSort*	n^2	n	n	1
Швидке (з рекурсією) *быстрая (сортировка Хоара) QuickSort (HoarSort)*	n^2	$n \cdot \log(n)$		n
Швидке (з ітераціями) *быстрая (сортировка Хоара) QuickSort (HoarSort)*	n^2	$n \cdot \log(n)$		log(n)
Включенням (вставлянням, простим включенням) /*включением/ InsertSort*	n^2		n	1
Шелла (включенням з убуваючим приростом) /*Шелла/ ShellSort*	n^2	$n^{1,25}$ або ($n^{1,5}$)	n	1
Екстракцією /*извлечением (простым извлечением)/ ExtractSort*	n^2			1
Деревом /*древесная (древовидная, с помощью бинарного дерева)/ TreeSort*	$n \cdot \log(n)$			n+2, де n ключів, 2 вказівника
Деревом з вибором із заміщенням	$n \cdot \log(n)$			$2 \cdot n - 1$
Пірамідальна (турнірна) /*пирамидальная/ HeapSort*	$n \cdot \log(n)$		$n \cdot \log(n)/2$	1
Розподіленням /*распределяющая (распределением, общая поразрядная, байтовая, цифровая поразрядная)/ RadixSort*	$k \cdot n$, або $(k \cdot n + s)$, де k–const, s–const			n (для масивів) log(n) (для списків)
Злиттям /*слиянием/ MergeSort*	$n \cdot \log(n)$			n

Пошук нового номеру (нового елементу) *NewNumbSort*	n^2	n	масив n
Кишенькова /*карманная, по ключу*/ *BinSort* (m–кількість ключів)	n + m	n ($n \geq m$)	1
Послідовного пошуку мінімумів *NextMinSearchSort*	n^2	n	1

Аналіз даних з таблиць дозволяє зробити ряд висновків.

На невеликих наборах даних ***доцільно*** використовувати ***сортування включенням***, оскільки зі всіх методів, що мають дуже просту програмну реалізацію, цей на практиці виявляється найшвидшим і ***при розмірностях <3000*** дає цілком прийнятну для більшості випадків швидкість роботи. Ще одна перевага цього методу полягає у тому, що він використовує повну або часткову впорядкованість вхідних даних і на впорядкованих даних працює швидше, а на практиці дані, як правило, вже мають хоча б частковий порядок.

Алгоритм ***бульбашкового сортування***, причому в тій його модифікації, яка не використовує частковий порядок даних початкового масиву, хоча і часто використовується, ***має погані показники*** навіть серед простих методів з квадратичною складністю.

Сортування Шелла виявляється лише красивим теоретичним методом, тому що на практиці використовувати його недоцільно: він складний в реалізації, але не дає такої швидкості, яку дають у порівнянні з ним по складності програмної реалізації методи.

При сортуванні великих масивів початкових даних краще використовувати ***швидке сортування.***

Якщо ж додається вимога гарантувати прийнятний час роботи швидке сортування у гіршому разі має складність $T(n^2)$, (хоча вірогідність такого випадку дуже мала), то треба застосовувати ***сортування деревом*** або ***сортування злиттям***. Як видно з таблиць, сортування злиттям працює ***швидше***, але слід пам’ятати, що воно вимагає додаткову пам’ять розміром порядку ***n.***

У тих же випадках, коли можливо дозволити використовувати додаткову пам’ять розміром порядку ***n***, має сенс скористатися ***сортуванням розподілом***.

Контрольні запитання і завдання для самоперевірки

1. Дайте визначення процесу сортування.

2. Які методи сортування Вам відомі ?
3. Охарактеризуйте та наведіть приклад внутрішнього сортування.
4. Охарактеризуйте та наведіть приклад зовнішнього сортування.
5. Порівняйте особливості зовнішнього і внутрішнього сортування.
6. Як оцінюється складність алгоритмів сортування ?
7. Яким чином можна поміняти значення пари змінних, наприклад, Y та U місцями?
8. Охарактеризуйте та наведіть приклад бульбашкового сортування з просіванням.
9. Охарактеризуйте та наведіть приклад сортування вибором.
10. Охарактеризуйте та наведіть приклад сортування включеннями.
11. Охарактеризуйте та наведіть приклад бульбашкового сортування вставками.
12. Охарактеризуйте та наведіть приклад сортування методом знаходження мінімального елементу.
13. Охарактеризуйте та наведіть приклад сортування методом знаходження нового елементу.
14. Охарактеризуйте та наведіть приклад швидкого сортування (сортування Хоара).
15. Охарактеризуйте та наведіть приклад сортування Шелла.
16. Охарактеризуйте та наведіть приклад шейкер-сортування.
17. Охарактеризуйте та наведіть приклад сортування деревом.
18. Охарактеризуйте та наведіть приклад пірамідального сортування (турнірного сортування).
19. Охарактеризуйте та наведіть приклад сортування із злиттям.
20. Охарактеризуйте та наведіть приклад порозрядного цифрового сортування.
21. Охарактеризуйте та наведіть приклад модифікації бульбашкового сортування з ознакою.
22. Охарактеризуйте та наведіть приклад бульбашкового сортування із запам’ятовуванням місця останньої перестановки.
23. Охарактеризуйте та наведіть приклад модифікації простими вставками.

Тестові завдання для самоконтролю

*1. Нехай елементи одновимірного масиву r [1:10] набувають відповідно значень –5; –3; –1; 1; 3; 5; 7; 9; 11;13. Які значення буде надруковано в результаті виконання дій: for i: =5 downto 1 do writeln(r[2*i]);?*

a) -5, -1,3, 7, 11;
b) 13, 9, 5, 1, -3;
c) -3, 1, 5, 9, 13;
d) -5, -3, -1, 1, 3;
e) 13, 11,9, 7,5.

2. Індексом у масиві називається:
a) значення елемента в масиві;
b) послідовність елементів одного типу;
c) структура з прямим доступом;
d) місцезнаходження елемента в масиві;
e) ім'я елемента масиву.

3. Опис масиву n:array [1..10] of real; означає:
a) масив N містить 11 дійсних чисел з індексами від 0 до 10;
b) масив N містить 10 цілих чисел з індексами від 1 до 10;
c) масив N містить 11 цілих чисел з індексами від 0 до 10;
d) масив N містить 10 чисел від 1 до 10;
e) масив N містить 10 дійсних чисел з індексами від 1 до 10.

4. Де допущено помилки у виведенні значення елемента масиву ?
a) write(‘ a [‘ , i ,’] = ‘ , a[i]) ;
b) write(‘ a [‘ , i ‘ ,] = ‘ , a[i]) ;
c) write(‘ a [‘ , j ,’] = ‘ , a[i]) ;
d) writeln(‘ a [‘ , i ,’ , ‘ , j , ‘] = ‘ , a[i,j]) ;
e) writeln(‘ a[i,j] ‘) .

5. Які методи впорядкування табличних величин Ви знаєте ?
a) метод заміни;
b) метод включення;
c) метод виключення;
d) метод обміну («бульбашки»);
e) метод вибору.

6. Яку максимальну кількість символів може містити рядкова величина ?
a) 255;
b) 254;
c) 256;
d) 250;
e) 512.

7. Нехай задана рядкова величина a: =‘ інформатика’. Який фрагмент цього тексту виділить процедура copy (a, 3,5) ?

a) рма;
b) інфор;
c) ика;
d) форма;
e) атика.

8. var b:string; b:= ‘ Алгоритм найкращий ‘; n: =length(b) ; Чому дорівнює n ?

a) 17;
b) 19;
c) 18;
d) 20;
e) 255.

9. Нехай змінна r має значення «Вінницький національний аграрний університет ». Чому дорівнює r після виконання процедури delete (r,1,10) ;?

a) аграрний ;
b) Вінницький;
c) національний аграрний університет;
d) національний;
e) Вінницький н.

10. Локальними називаються змінні, що:

a) описані в допоміжних алгоритмах;
b) описані в основному алгоритмі;
c) описані у виведенні результатів;
d) описані в підпрограмах;
e) описані в заголовку.

11. Параметри, які передаються у підпрограму при її виклику, називаються:

a) формальними;
b) фактичними;
c) глобальними;
d) локальними;
e) функціональними.

12. Опис підпрограм здійснюють за допомогою службових слів:

a) insert;

b) function;
c) procedure;
d) delete;
e) begin.

13. Задайте зелений колір майбутнього зображення:
a) setbkcolor(3);
b) setcolor (green) ;
c) putpixel(100,100,2) ;
d) putpixel(100,100,3);
e) setcolor(2).

14. Яка процедура ініціалізує графічний режим?
a) driver;
b) mode;
c) pieslice;
d) initgraph;
e) detect.

15. Запишіть команду малювання відрізка з кінцями (150,150), (200, 200) поточним кольором).
a) lineto(150,200) ;
b) line(150,200,150,200) ;
c) 1ineto(150,150,200,200) ;
d) line(150,150,200,200);
e) 1ine(150,200).

16. Що робить команда closegraph ?
a) закриває графічний режим;
b) задає спосіб заповнення замкнутої області;
c) задає графічний режим;
d) відкриває графічний режим;
e) виводить текст у графічному режимі;

17. Функція визначення кількості елементів у типізованому файлі:
a) seek ;
b) filesize;
c) append;
d) read;
e) eoln.

18. Процедура відкриття файла даних для читання:

a) rewrite;
b) close;
c) erase;
d) reset;
e) rename.

19. Вкажіть, де допущені помилки при введенні даних?
a) readln (‘ Значення рівне ‘ , b: 2 : 4) ;
b) read (b: 2:4) ;
c) readln (b, m) ;
d) read (b;m) ;
e) readln (a,b,c) ;

*20. Нехай елементи одновимірного масиву r [1:10] набувають відповідно значень -5; -3; -1; 1; 3; 5; 7; 9; 11; 13. Які значення буде надруковано в результаті виконання дій: for i: =5 downto 1 do writeln(r[2*i–l]);?*
a) -5, -1, 3, 7, 11;
b) 13, 9, 5, 1, -3;
c) -3, 1, 5, 9, 13;
d) -5, -3, -1, 1, 3;
e) 11, 7, 3, -1, -5.

22. Опис масиву n: array [1. . 15] of integer; означає:
a) масив N містить 16 цілих чисел з індексами від 0 до 15;
b) масив N містить 15 дійсних чисел з індексами від 1 до 15;
c) масив N містить 15 цілих чисел з індексами від 1 до 15;
d) масив N містить 11 цілих чисел з індексами від 0 до 10;
e) масив N містить 15 чисел від 1 до 15.

23. Де допущені помилки у виведенні значення елемента масиву ?
a) write(‘ b [‘ , i ,’] = ‘ , b[i]) ;
b) write(‘ b [‘ , i ‘ ,] = ‘ , b[i]) ;
c) writeln(‘ b [‘ , i ,’ , ‘ , j , ‘] = ‘ , b[i,j]) ;
d) write(‘ b [‘ , j ,’] = ‘ , b[i]) ;
e) writeln(‘ b [i,j] ‘) ;

24. Яких методів впорядкування табличних величин не існує ?
a) метод заміни;
b) метод включення;
c) метод виключення;
d) метод обміну («бульбашки»);

e) метод вибору.

25. Яку максимальну кількість символів може містити рядкова величина ?

a) 254;
b) 256;
c) 255;
d) –255;
e) 300.

РОЗДІЛ 3
ПОШУК ЕЛЕМЕНТІВ У МАСИВАХ

План (логіка) викладу матеріалу:

3.1. Пошук перебором (лінійний пошук)
3.2. Лінійний пошук з бар'єром
3.3. Бінарний пошук
3.4. Пошук Фібоначчі
3.5. Інтерполяційний пошук елементу в масиві
3.6. Пошук з перестановкою в початок
3.7. Пошук з транспозицією

Інший клас дуже важливих та часто використовуваних алгоритмів складають алгоритми, що реалізують різноманітні методи пошуку заданого елемента серед множини компонент визначеної структури даних [1, 22, 26].

В умовах розвитку ринкової економіки, посилення конкурентної боротьби, розвитку міжнародних інтеграційних відносин в Україні виникає потреба впровадження в управління економічними процесами економіко-математичних методів моделювання з використанням сучасних інформаційних технологій і комп'ютерних засобів, їх оптимізації [34, 35]. Оптимізуючи пошук елементів у масивах можливо збільшити ефективність економіко-математичних методів моделювання.

Методи пошуку, як і методи сортування розділяють на медоди пошуку в основній пам'яті та методи пошуку у зовнішній пам'яті (внутрішні та зовнішні).

Не дивлячись на використання того ж набору термінів (дерева і хешування), пошук в зовнішній пам'яті і критерії оцінки ефективності алгоритмів істотно відрізняються від тих, які застосовні у разі розташування даних в основній пам'яті.

Розглянемо структури даних в основній пам'яті і методи їх використання, що призначені для пошуку даних відповідно до значень їх ключів. Ця задача є не менш поширеною в програмуванні, ніж внутрішнє сортування даних.

Головним чином, поширені два підходи – пошук в динамічних деревовидних структурах і пошук в таблицях на основі хешування.

До основних методів пошуку у масивах (у основній пам'яті) відносять:

1. Пошук перебором (лінійний пошук).
2. Лінійний пошук з бар'єром.
3. Бінарний пошук.

4. Пошук Фібоначчі.
5. Інтерполяційний пошук елементу в масиві.
6. Пошук з перестановкою в початок.
7. Пошук з транспозицією.

3.1. Пошук перебором (лінійний пошук)

Полягає в послідовному переборі всіх елементів початкового масиву і пошуку якнайменшого індексу ***i*** компоненти масиву із значенням ***x***.

Щоб знайти дані у неврегульованому масиві, застосовується **алгоритм *простого лінійного перебору елементів***. Наведена функція повертає індекс заданого елементу масиву. Її аргументи: масив з першим індексом ***0***, кількість елементів в масиві та шукане число. Якщо число не знайдено, повертається ***-1***.

Функція пошуку простим лінійним перебором елементів:

```
function SearchPer (Arr : array of Integer;
        n, v : Integer) : Integer;
var  i : Integer;
begin
 Result := -1;
 for i := 1 to n do
 if Arr [i] = v then begin
  Result := i; Exit;
 end;
end;
```

Програма пошуку простим перебором:

```
PROGRAM Find_1 (Input,Output); {лінійний пошук}
 const N = 8;
 type R = 0..N;
 var i: R;
   Q: Boolean;
   A: Array [1..N] of Integer;
   X: Integer;
BEGIN
  WriteLn ('початковий масив...');
```

```
For i:=1 to N do A[i]:=Random(12);
For i:=1 to N do Write (A[i],' ');
WriteLn;
Write ('Вкажіть елемент, який шукатимете: ');
ReadLn (X);
i:=0;
Repeat
 i:=i+1; Q:=(A[i]=X)
until Q OR (i=N);
If Q then
begin
WriteLn ('Елемент, що цікавить, знайдено;');
Write ('вперше він зустрічається в масиві');
 WriteLn (' під номером ',i);
 end
 else
 WriteLn('Елемент, що цікавить,
                                        не знайдено!');
END.
```

3.2. Лінійний пошук з бар'єром

У кінець масиву додається шуканий елемент (бар'єр). При пошуку завжди буде знайдений індекс ***i*** компоненти масиву із значенням ***x,*** але якщо значення ***i*** більше довжини початкового масиву, то шуканий елемент вважається ненайденим.

Програма лінійного пошуку з бар'єром:

```
PROGRAM Find_2 (Input,Output);
                          {Лінійний пошук з бар'єром}
const N = 7;
  N1 = 8;        {N:=N+1}
type R = 0..N1;
 var i : R;
  Q : Boolean;
  A : Array [1..N1] of Integer;
  X : Integer;
BEGIN
WriteLn ('вхідний масив...');
```

```
For i:=1 to N do A[i] := Random(12);
For i:=1 to N do Write (A[i],' ');
WriteLn;
WriteLn ('Вкажіть елемент,
                                    який шукатимете:');
ReadLn (X); i:=0; A[N+1]:=X;
Repeat i:=i+1 until A[i]=x;
If i<>N+1 then
begin
 WriteLn('Елемент, що цікавить, знайдено;');
 Write('вперше він зустрічається в масиві');
 WriteLn (' під номером ',i);
end
else
WriteLn('Елемент, що цікавить,
                                        не знайдено!');
END.
```

3.3. Бінарний пошук

Основна ідея бінарного пошук (або пошуку діленням навпіл) полягає у випадковому обиранню деякого елементу початкового ***відсортованого*** масиву з індексом ***i***, та порівнянні його з аргументом пошуку ***x***. Якщо він дорівнює ***x***, то пошук закінчується, якщо він менше ***x***, то робиться висновок, що всі елементи з індексами, меншими або рівними ***i***, можна виключити з подальшого пошуку; якщо ж він більше ***x***, то виключаються індекси більші та рівні ***i***. Вибір ***m*** абсолютно не впливає на коректність алгоритму, але впливає на його ефективність. Очевидно, що чим більша кількість елементів виключається на кожному кроці алгоритму, тим цей алгоритм ефективніший. Оптимальним рішенням буде вибір середнього елементу, оскільки при цьому у будь-якому випадку виключатиметься половина масиву.

Бінарний пошук можна використовувати тільки тоді, коли компоненти масиву впорядковані !

Приклад функції бінарного пошуку наведено нижче. На початку змінна ***Up*** вказує на найменший елемент масиву ***(Up := 0), Down*** – на найбільший (***Down := n***, де ***n*** – верхній індекс масиву), ***Mid*** – на середній. Далі, якщо шукане число рівне ***Mid***, то задача вирішена; якщо число менше ***Mid***, то потрібний нам елемент лежить нижче середнього, і за нове значення ***Up*** приймається ***Mid+1***; якщо потрібне нам число менше середнього елементу, значить, воно розташоване вище середнього елементу, і ***Down:= Mid–1***. Потім слідує нова ітерація циклу, і так

повторюється до тих пір, доки не знайдеться потрібне число, або ***Up*** не стане більше ***Doun.***

Функція бінарного пошуку:

```
function SearchBin (Arr : array of Integer;
     v, n : Integer) : Integer;
var Up, Down, Mid : Integer;
    Found : Boolean;
begin
 Up:=0; Down:=n; Found:=False; Result:=-1;
 repeat
 Mid := Trunc ((Down - Up) / 2) + Up;
 if Arr [Mid] = v then Found := True
 else
  if v < Arr [Mid] then Down := Mid - 1
   else Up := Mid + 1;
 until (Up > Down) or Found;
 if Found then Result := Mid;
end;
```

Приклад програми бінарного пошуку:

```
PROGRAM Find_3 (Input,Output);
  Const  N = 8;
   var i,j,k : Integer;
    Q : Boolean;
    X : Integer;
    A : Array [1..N] of Integer;
  BEGIN
  WriteLn ('Ввести елемент масиву...');
  For i:=1 to N do ReadLn (A[i]);
  WriteLn ('Ось введений масив...');
  For i:=1 to N do Write (A[i],' ');
  WriteLn;
  WriteLn ('Вкажіть елемент,
                                    який шукатимете...');
  ReadLn (X);
  i:=1; j:=N; Q:=FALSE;
  Repeat
   k:=(i+j)DIV 2;
   If A[k]=X then Q:=TRUE else
```

```
    If A[k]<X then i:=k+1 else j:=k-1
  until Q OR (i>j);
  If Q then
  begin
   WriteLn('Елемент, що цікавить, знайдено;');
   Write('вперше він зустрічається в масиві');
   WriteLn (' під номером ',i);
  end
  else
  WriteLn('Елемент,що цікавить,не знайдено!');
  END.
```

Програма бінарного пошуку індексу компоненти ***масиву A[1..N]*** із значенням ***Key*** (рекурсивний варіант).

Приклад програми бінарного пошуку (рекурсивний варіант):

```
PROGRAM Find_4 (Input,Output);
  const N = 8; {Кількість елементів у масиві}
  type Massiv = Array [1..N] of Integer;
  var A : Massiv;
    {Масив, відсортований за збільшенням}
   Key : Integer;   {Шуканий елемент}
  Search : 1..N; {Індекс знайденого елементу}
      {0, якщо елемент не знайдено}
   i : Integer;
  Low,High : Integer;
 {--------------------}
 FUNCTION Bin_Search( A : Massiv;
     Low : Integer;
     High : Integer;
      X : Integer):Integer;
 var Mid: Integer;
 BEGIN
 If Low>High then Bin_Search:= 0
 else
 begin
  Mid := (Low+High) DIV 2;
  If X=A[Mid] then
   Bin_Search:=Mid
  else
   If X<A[Mid] then
   Bin_Search:=Bin_Search(A,Low,Mid-1,X)
```

```
   else
   Bin_Search:=Bin_Search(A,Mid+1,High,X);
  End;
 END;
 {--------------------}
BEGIN
  WriteLn ('Введіть масив, відсортований
                                    за збільшенням...');
  For i:=1 to N do
  begin
  Read (A[i]);
  Write (' ')
  end;
  Writeln;
  Write ('Введіть шуканий елемент... ');
  ReadLn (Key);
  WriteLn (Bin_Search(A,1,N,Key));
END.
```

3.4. Пошук Фібоначчі

У дереві Фібоначчі числа в дочірніх вузлах, відрізняються від числа в батьківському вузлі на одну і ту ж величину, а саме на число Фібоначчі (***a[1]=1,a[2]=1,a[n]=u[n–1]+u[n–2], n>2, наприклад: 1, 1, 2, 3, 5, 8, 13, 21,*** **…**). Суть методу у тому, що порівнюючи наше шукане значення з черговим значенням в масиві, ми не ділимо навпіл нову зону пошуку, як в бінарному пошуку, а відступаємо від попереднього значення, з яким порівнювали, в потрібну сторону на число Фібоначчі.

Програма алгоритму пошуку Фібоначчі у масиві відсортованому за збільшенням:

```
PROGRAM Find_5 (Input,Output);
 const  N = 8;  {Кількість елементів в масиві}
 type Massiv = Array [1..N] of Integer;
 var A : Massiv;
                  {Масив, відсортований за збільшенням}
   Key : Integer;   {Шуканий елемент}
  Search : Integer; {Індекс знайденого елементу}
      {0, якщо елемент не знайдений}
   Mid : Integer;
                          {Індекс внутрішнього елементу}
  j,i : Integer;
```

```
 F1,F2,t: Integer;
 Finish : Boolean;
{-----------------}
FUNCTION Fib (N: Integer): Integer;
 var F1,F2 : Integer;
BEGIN
If (N=0) OR (N=1) then Fib:= N else
 If N>=2 then
 begin
  F1 := Fib(N-1);
  F2 := Fib(N-2);
  Fib:= F1 + F2;
 End;
END;
{-----------------}
BEGIN
 WriteLn ('Введіть відсортований масив... ');
 For i:=1 to N do
 begin
 Read (A[i]); Write (' ')
 end;
 WriteLn;
 Write('Введіть шуканий елемент... ');
 ReadLn (Key);
 j := 1;
 While Fib (j)<N+1 do
 begin
 j := j+1;
 Mid := N-Fib(j-2)+1;
 F1 := Fib(j-2);
 F2 := Fib(j-3);
 Finish := FALSE;
 While (Key<>A[Mid]) AND (Finish=FALSE) do
  If (Mid<=0) OR (Key>A[Mid]) then
  If F1=1 then Finish := TRUE else
  begin
   Mid := Mid+F2;
  F1 := F1-F2;
  F2 := F2-F1;
  end;
 else
  If F2=0 then Finish := TRUE else
  begin
```

```
   Mid := Mid-F2;
   t := F1-F2;
   F1 := F2;
   F2 := t;
   end;
  If Finish then Search := 0
  else Search := Mid;
 end;
 WriteLn('Індекс шуканого елементу
                                  у масиві... ',Search)
END.
```

3.5. Інтерполяційний пошук елементу в масиві

Уявимо ситуацію, коли шукається слово в словнику. Маловірогідно, що спочатку відкриємо середину словника, потім відступимо від початку на 1/4 або 3/4 і т.ін. (як в бінарному пошуку). Якщо потрібне слово починається з літери ***'A'***, то напевно пошук почнемо десь з початку словника (тобто знайдена відправна точка для пошуку).

Якщо помітимо, що шукане слово має знаходитися набагато далі відкритої сторінки, то пропустимо велику кількість сторінок, перш ніж зробити нову спробу. Це значно відрізняється від попередніх алгоритмів, що не роблять різниці між *'багато більше'* і *'трохи більше'*.

Суть алгоритму інтерполяційного пошуку: якщо відомо, що ***K*** лежить між ***Kl*** и ***Ku***, то наступну спробу робимо на відстані ***(u–l)(K–Kl)/(Ku–Kl)*** від ***l***, припускаючи, що ключі є числами, що зростають приблизно в арифметичній прогресії.

Фрагмент програми з використанням інтерполяційного методу пошуку:

```
{ Інтерполяційний пошук в масиві K[1..N] числа X }
…
l:=1;
u:=N;
while u>=l do
begin
 i:=l+(u-l)*(X-K[l])/(K[u]-K[l]);
 if X<K[i] then u:=i-1
  else if X>K[i]
   then l:=i+1
    else
```

```
    begin
    writeln('ЗНАЙШЛИ');
    X:=K[i];
    end;
end;
writeln('НЕ ЗНАЙШЛИ');

...
```

Інтерполяційний пошук працює за ***log log N*** операцій, якщо дані розподілені рівномірно. Як правило, він використовується лише на дуже великих таблицях, причому робиться декілька кроків інтерполяційного пошуку, а потім на малому підмасиві використовується бінарний або послідовний варіанти.

3.6. Пошук з перестановкою в початок

Це лінійний пошук, у якому знайдений елемент переміщується в початок масиву. Таким чином даний спосіб дає виграш за часом в тому випадку, якщо часто виконується пошук елементів, що повторюються.

3.7. Пошук з транспозицією

Це лінійний пошук, в якому знайдений елемент переставляється на один елемент до голови списку. І якщо до цього елементу звертаються часто, то, переміщуючись до голови списку, він скоро виявиться на першому місці.

Додаткові відомості про цей метод. Для виконання завдання масив зручно представити у вигляді лінійки елементів, в яких зберігаються числа. Наприклад:

Масив ABC

-4	16	12	0	-77	11	32	77	-41	2	12	–98	91	значення елементів
1	2	3	4	5	6	7	8	9	10	11	12	13	індекси

При сортуванні, наприклад, за зростанням, можна обрати два методи.

Метод 1.

Знайти в масиві ***ABC*** найменший елемент та його номер і встановити його на ***1***-ше місце, а ***1***-й елемент – на місце, де був знайдений найменший елемент. У даному прикладі найменшим є ***12***-й елемент, а його значення дорівнює ***98***.

Після перестановки масив буде мати такий вигляд:

Масив ABC

-98	16	12	0	-77	11	32	77	-41	2	12	-4	91	значення елементів
1	2	3	4	5	6	7	8	9	10	11	12	13	індекси

Тепер аналогічні дії можна повторити, але пошук мінімального і перестановку треба почати з другого елементу. В даному випадку мінімальним буде п’ятий елемент із значенням ***-77***. Після перестановки масив буде мати такий вигляд :

Масив ABC

-98	-77	12	0	16	11	32	77	-41	2	12	-4	91	значення елементів
1	2	3	4	5	6	7	8	9	10	11	12	13	індекси

Таким чином, описані дії треба повторювати доти, доки не буде досягнуто ***12-го*** елементу. Після цього масив ***ABC*** буде відсортовано за зростанням.

Отже, якщо розмірність масиву дорівнює ***N***, то треба створити два вкладених цикли, де зовнішній має ***N–1*** разів повторити внутрішній. У внутрішньому циклі треба виконувати пошук найменшого елементу і його індекса.

При цьому, після закінчення внутрішнього циклу треба виконувати дві дії:

– робити перестановку знайденого мінімального елементу з елементом, що знаходится на початку чергового пошуку;

– збільшувати на ***1*** (одиницю) спеціальну змінну ***BEG***, яка має вказувати на початковий номер елементу, з якого наступного разу буде відбуватись пошук мінімального елементу.

Приклад фрагменту програми:

```
BEG:=1; {починаємо шукати з 1 елемента : }
for i:=BEG to 12 do
begin
 Num_Min:=BEG;
```

```
                {припустимо номер мін.елемента = BEG}
    Min:=ABC[Num_Min];  {а це його значення}
    for k:=BEG to 12 do
    if {.......}
    begin
        {тут відшукаємо номер найменшого елементу в }
         {массиві в діапазоні індексів від BEG до 12}
    end; {if}
       {переставимо значення елемента BEG та Num_Min}
    Z:=ABC[BEG];
     {запам'ятаємо значення початкового елемента в Z}
    ABC[BEG]:=ABC[Num_Min];
                   {перепишемо значення MIN в ел. BEG}
    ABS[Mun_Min]:=Z;
                        {а значення ел.BEG – в Num_Min}
    BEG:=BEG+1;
              {зсунемо початок наступного пошуку на 1}
    end; {for}
```

Метод 2

Полягає в тому, що елементи масиву порівнюються *послідовно* та *попарно*, і якщо наступний буде меншим за попередні, то вони міняються місцями. Тобто більший з двох елементів «спливає» – просувається в бік зростання значень масиву. Тому цей метод називається бульбашковим. Робота починається з першого та другого елементів. У даному випадку перший елемент менший ніж другий, отже, перестановку не виконуємо. Після цього порівнюємо другий і третій елементи. Треба робити перестановку. Тепер масив має такий вигляд:

Масив ABC

-4	12	16	0	-77	11	32	77	-41	2	12	-98	91	значення елементів
1	2	3	4	5	6	7	8	9	10	11	12	13	індекси

Для третього і четвертого елементів дії аналогічні.

Масив ABC

-4	12	0	16	-77	11	32	77	-41	2	12	-98	91	значення елементів
1	2	3	4	5	6	7	8	9	10	11	12	13	індекси

Ці дії треба повторювати до порівняння ***12*** та ***13*** елементів, а потім цю всю процедуру (спочатку) треба повторювати ще ***11*** разів.

У випадку, якщо масив має розмірність ***N***, треба створити ***2*** вкладених цикли з кількістю проходів ***N-1*** кожний. У внутрішньому циклі будуть порівнюватись та переставлятись за необхідністю сусідні елементи, а зовнішній має його повторити ***N-1*** разів.

Контрольні запитання і завдання для самоперевірки

1. Охарактеризуйте та наведіть приклад пошуку перебором (лінійний пошук).
2. Охарактеризуйте та наведіть приклад лінійного пошуку з бар'єром.
3. Охарактеризуйте та наведіть приклад бінарного пошуку.
4. Охарактеризуйте та наведіть приклад пошуку Фібоначчів.
5. Охарактеризуйте та наведіть приклад інтерполяційного пошуку елементу в масиві.
6. Охарактеризуйте та наведіть приклад пошуку з перестановкою в початок.
7. Охарактеризуйте та наведіть приклад пошуку з транспозицією.
8. Дайте визначення бінарного дерева. Наведіть приклади.
9. Опишіть методи проходження дерева.
10. Який порядок проходження дерева при послідовному методі ?
11. Порівняйте переваги та недоліки методів пошуку.

Тестові завдання для самоконтролю

1. x: =4567. Яке значення y буде отримано в результаті виконання команди: y:=x div 100 ?

a) 45.67;
b) 0.4567;
c) 45;
d) 4.5E1;
e) 45.0.

2. x: =4567. Яке значення y буде отримано в результаті виконання команди: y:=x mod 100 ?

a) 67E–2;
b) 0.67;
c) 45E–2;
d) 67;
e) 67.0.

3. Де є помилки в записах функцій ?

a) cos (4*x) ;
b) abc(7*x) ; c)cos*2x; d)sin(3*x); e)cos(3*x)A2;

10. Яка функція знаходить квадрат виразу ?

a) sqr;
b) sqrt;
c) exp;
d) abs;
e) ln.

4. Як знайти котангенс змінної m ?

a) ctan (m) ;
b) cos/sin m;
c) ctg (m) ;
d) cosm/sinm;
e) cos (m) /sin (m) ;

*5. a:=1; b:=–5; c:=6. Якого значення набуде величина d, якщо d:=sqrt(sqr(b)–4*a*c) ?*

a) l;
b) 49;
c) -7;
d) -l;
e) 7.

6. Як знайти квадрат із суми коренів x та y ?

a) sqrt(x*x+y*y);
b) sqrt(sqr(x)+sqr(y));
c) (sqrt(x)+sqrt(y))*(sqrt(x)+sqrt(y));
d) sqr(sqrt(x)+sqrt(y));
e) sqr(sqr(x)+y*y)) ;

7. Якого значення набуде змінна y в результаті виконання дій: a:=9; if a>7 then y:=–1 else y:=10 ?

a) 11
b) 1;
c) 10;
d) 5;
e) 8.

8. Якого значення набуде змінна y в результаті виконання дій: a:=9; if (a>7) or (a<6) then y:=–1 else y:=10 ?

a) 5;
b) 6;
c) -1;
d) 8;
e) 10.

9. Якого значення набуде змінна y в результаті виконання дій: a:=9; if (a>7) and (a<6) then y:=–l else y:=10?

a) -1;
b) 5;
c) 6;
d) 10;
e) 8.

*10. Якого значення набуде змінна m у результаті виконання дій: m:=l; for i:=3 to 6 do m:=m*i ;?*

a) 120;
b) 360;
c) 20;
d) 240;
e) 0.

11. Якого значення набуде змінна n у результаті виконання дій: n:=l; while n<10 do n:=n+4;?

a) 12;
b) 14;
c) 9;
d) 0;
e) 13.

12. Якого значення набуде змінна b в результаті виконання таких дій: b:=1; repeat b:=b/2; until b<0.5;?

a)l;
b) 0.5;

c) 0.25;
d) -0.25;
e) 0.

13. Дано масив:
Визначте результат виконання фрагмента програми:
P := 1;
*For i := 1 to 5 do If A[i]<=5 Then P := P*A[i];*
a) 15
b) 3
c) 18
d) 25

14. Дано масив:
Визначте результат виконання фрагмента програми:
S := 0;
For i := 1 to 5 do If i mod 2 =0 Then S := S+ A[i];
a) 15
b) 14
c) 16
d) 17

15. Дано масив:
Визначте результат виконання фрагмента програми:
S := 0;
For i := 1 to 5 do If A[i] <6 Then S := S + A[i];
a) 14
b) 8
c) 29
d) 17

16. Які задачі належать до задач на пошук у масиві елемента із заданою властивістю?
a) Знаходження суми додатних елементів масиву
b) Визначення кількості додатних елементів массиву
c) Визначення найбільшого елемента массиву
d) Подвоєння значень елементів масиву

17. Які задачі належать до задач на змінювання значень елементів масиву?
a) Знаходження суми додатних елементів масиву
b) Заміна від’ємних елементів масиву нулем

c) Визначення найбільшого елемента массиву
d) Подвоєння значень елементів масиву

18. Встановіть відповідність між операторами та їх призначенням.

a) Подвоєння значень елементів массиву
b) Оголошення масиву 10 дійсних чисел
c) Заповнення масиву випадковими числами в діапазоні [5; 14]
d) Знаходження суми елементів масиву
__ For i := 1 to 10 do S := S + A[i]
__ For i := 1 to 10 do A[i] := Random(10)+5
__ var A: array[1..10] of Real
__ For i := 1 to 10 do A[i] := A[i]*2

19. Встановіть відповідність між операторами та їх призначенням.

a) Знаходження суми від'ємних елементів масиву
b) Заміна нулями від'ємних елементів массиву
c) Визначення кількості від'ємних елементів массиву
d) Виведення від'ємних елементів масиву у поле списку ListBox1
1) __ For i := 1 to 10 do If A[i]<0 Then S := S + A[i]
2) __ K := 0; For i := 1 to 10 do If A[i]<0 Then K := K + 1
3) __ For i := 1 to 10 do If A[i]<0 Then ListBox1.Items.Add (IntToStr(A[i]))
4) __ For i := 1 to 10 do If A[i]<0 Then A[i]:= 0

20. Дано програму:
var a: array [1..8] of Integer; M, k: Integer;
begin {...} M := a[1];
For k := 2 to 8 do If M < a[k] Then M := a[k]; {...}
Скільки разів буде виконаний оператор M := a[k] для масиву (3, 8, 7, 9, 4, 10, 2, 12)?

21. Для кожної пари сусідніх елементів масиву A[1..6] виконується операція
S := S + Byte (A[i] > A[i+1]) (Byte (True) = 1; Byte (False) = 0)
Початкове значення S дорівнює 0. Чому дорівнює кінцеве значення S, якщо вхідний масив було впорядковано за зростанням?

a) 5
b) Не можна визначити
c) 6
d) 0

22. Які існують способи сортування масиву?

a) За зростанням

b) За прискоренням

c) За неспаданням

d) За спаданням

23. Установіть правильний порядок операторів, що реалізують упорядкування масиву методом вибору максимального елемента.

Вкажіть послідовність всіх 5 варіантів відповіді:

__ For K := 10 downto 2 do

__ For i := 2 to K do

__ C := X[M]; X[M] := X[K]; X[K] := C; end;

__ If [Xi] > Max Then begin Max := X[i]; M := i; end;

__ begin M := 1; Max := X[1];

24. Чому дорівнює A після виконання фрагмента програми:

```
var mas: array[1..10] of Real; A: Real;
begin
mas[1]:=14;
mas[5]:=3;
mas[9]:=8;
A:=(mas[9] – mas[1])*mas[5];
{...}
```

25. Установіть правильний порядок операторів, що реалізують упорядкування масиву методом бульбашки.

Вкажіть послідовність всіх 5 варіантів відповіді:

__ Repeat Prap := False;

__ For i := 1 to 9 do

__ Until Prap = False

__ C := X[i]; X[i] := X[i+1]; X[i+1] := C; Prap := True end

__ If X[i] > X[i+1] Then begin

КОМП'ЮТЕРНИЙ ПРАКТИКУМ ДЛЯ САМОСТІЙНОЇ РОБОТИ

ЗАВДАННЯ

1. Розробити граф-схеми алгоритмів згідно варіанту (частина 1 та частина 2).
2. Написати програму за отриманими граф-схемами алгоритмів.
3. Провести спостереження процесу сортування та пошуку за допомогою налагоджувача в режимі DEBUGGER. Результати показати викладачеві.
4. Забезпечити виведення на екран вхідних та вихідних масив у вигляді стовпчика, де в кожному рядку виводиться назва, індекс та значення елементу масива. Наприклад, U[12]= 69.
5. Оформити звіт та захистити роботу.

Частина 1. Методи сортування

Варіанти методів сортування

Номер методу	Метод сортування
1	Сортування методом «бульбашки»
2	Бульбашкове сортування з просіванням
3	Сортування вибором
4	Сортування включеннями
5	Бульбашкове сортування вставками
6	Сортування методом знаходження мінімального елементу
7	Сортування методом знаходження нового елементу
8	Сортування Хоара рекурсивний алгоритм
9	Сортування Хоара нерекурсивний алгоритм
10	Сортування Шелла
11	Шейкер-сортування
12	Метод послідовного пошуку мінімумів
13	Сортування із злиттям
14	Сортування деревом
15	Пірамідальне сортування
16	Порозрядне цифрове сортування

Варіанти завдань

Варі–ант	Завдання	Метод
1	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	1
2	Дано ***N*** цілих чисел. Впорядкувати їх за зростанням	2
3	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***.	3

Варі–ант	Завдання	Метод
	Відсортувати за зростанням	
4	Дано ***N*** цілих чисел від ***-50*** до ***50***. Впорядкувати їх за зростанням	4
5	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за зростанням	5
6	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за зменшенням	6
7	Дано ***N*** цілих чисел. Впорядкувати їх за убуванням	7
8	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за убуванням	8
9	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за убуванням	9
10	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за убуванням	10
11	Розташувати елементи одновимірного масиву ***A[1..n]*** залежно від значення параметра ***q*** або у порядку зростання, або у порядку зменшення	11
12	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	12
13	Дано ***N*** цілих чисел. Впорядкувати їх за зростанням	1
14	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за зростанням	2
15	Дано ***N*** цілих чисел від ***-50*** до ***50***. Впорядкувати їх за зростанням	3
16	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за зростанням	4
17	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за зменшенням	5
18	Дано ***N*** цілих чисел. Впорядкувати їх за убуванням	6
19	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за убуванням	7
20	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за убуванням	8
21	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за убуванням	9
22	Розташувати елементи одновимірного масиву ***A[1..n]*** залежно від значення параметра ***q*** або у порядку зростання, або у порядку убування	10
23	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	11
24	Дано ***N*** цілих чисел. Впорядкувати їх за зростанням	12
25	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за зростанням	1
26	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за зростанням	2
27	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за зростанням	3
28	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за зменшенням	4
29	Дано ***N*** цілих чисел. Впорядкувати їх за убуванням	5

Варі–ант	Завдання	Метод
30	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за убуванням	6
31	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за убуванням	7
32	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за убуванням	8
33	Розташувати елементи одновимірного масиву ***A[1..n]*** залежно від значення параметра ***q*** або у порядку зростання, або у порядку убування	9
34	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	10
35	Дано ***N*** цілих чисел. Впорядкувати їх за зростанням	11
36	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за зростанням	12
37	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за зростанням	1
38	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за зростанням	2
39	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за зменшенням	3
40	Дано ***N*** цілих чисел. Впорядкувати їх за убуванням	4
41	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за убуванням	5
42	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за убуванням	6
43	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за убуванням	7
44	Розташувати елементи одновимірного масиву ***A[1..n]*** залежно від значення параметра ***q*** або у порядку зростання, або у порядку убування	8
45	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	13
46	Дано ***N*** цілих чисел. Впорядкувати їх за зростанням	14
47	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за зростанням	15
48	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за зростанням	16
49	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати за зростанням	13
50	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за зменшенням	14
51	Дано ***N*** цілих чисел. Впорядкувати їх за убуванням	15
52	У масиві ***A[1..n]*** кожен елемент змінюється в межах від ***0*** до ***20***. Відсортувати за убуванням	16
53	Дано ***N*** цілих чисел від ***–50*** до ***50***. Впорядкувати їх за убуванням	13
54	У масиві ***A[1..30]*** кожен елемент дорівнює ***0, 7*** або ***14***. Відсортувати у порядку зменшення	14
55	Розташувати елементи одновимірного масиву ***A[1..n]*** залежно від значення параметра ***q*** або у порядку зростання, або у порядку	15

Варі–ант	Завдання	Метод
	зменшення	
56	У масиві ***A[1..n]*** кожен елемент дорівнює ***0, 1*** або ***2***. Відсортувати за збільшенням	16

Частина 2. Електронний довідник з рядковими масивами

1. Розробити програму електронного довідника деканату. Робочі поля - *прізвище, ім`я, по-батькові, місце проживання*, *дата народження* (у вигляді DD.MM.PP).

2. Розробити програму електронного довідника ДАІ. Робочі поля - *номерний знак, марка автомобіля, власник, місце проживання.*

3. Розробити програму електронного англо- (або німецько-, або фрацузько-) українського словника. Робочі поля - *слово на іноземній мові*, *його переклад, коментар або пояснення.*

4. Розробити програму електронного довідника з аналізу жирності молока, що поступає з господарства на молокозавод. Робочі поля – *назва господарства, прізвище шофера, марка автомобіля, кількість молока, жирність.*

5. Розробити програму електронного довідника для обліку продукції тваринництва. Робочі поля – *назва продукції, кількість, ціна, сума, рахунок обліку*

6. Розробити програму електронного довідника для відпущених зі складу сільськогосподарських продуктів. Робочі поля – *назва продукту (напр. Озима пшениця), кількість, ціна, сума, споживач.*

7. Розробити програму електронного бібліотечного каталогу. Робочі поля – *назва книги, автор, рік видання, класифікатор.*

8. Розробити програму електронного довідника для товарів магазину. Робочі поля - *назва товару, ціна, залишок, постачальник.*

9. Розробити програму електронного довідника для продуктів, що поступають з поля на склад. Робочі поля - *назва продукту, прізвище шофера, марка автомобіля, кількість продукту.*

10. Розробити програму електронного довідника з автобусних рейсів. Робочі поля - *напрямок (напр. "Літин"), номер рейсу, час відправлення, час прибуття в пункт призначення.*

11. Розробити програму електронного довідника для обліку приплоду телят. Робочі поля – *прізвище доярки, інвентарний номер корови, стать теляти, жива вага.*

12. Розробити програму електронного довідника для обліку основних фондів. Робочі поля – *назва основних фондів, рахунок обліку, сума на початок року, знос за рік, сума на кінець року.*

13. Розробити програму електронного телефонного довідника. Робочі поля - *номер телефону, прізвище, ім'я, по-батькові власника телефону, назва вулиці, номер будинку, номер квартири.*

14. Розробити програму електронного довідника розрахунку витрат для годівлі ВРХ. Робочі поля – *назва корму, кількість корму, ціна, сума витрат.*

15. Розробити програму електронного довідника використання пального на польових роботах. Робочі поля – *назва трактора, характер роботи, обсяг роботи, норма використання пального, фактично використано пального.*

16. Розробити програму електронного довідника обліку витрат в людино-днях на виробництво 1 ц продукції рослинництва. Робочі поля – *назва продукції, прізвище агронома, номер поля, витрати в людино-днях.*

17. Розробити програму електронного довідника обліку внесення добрив у грунт. Робочі поля – *номер бригади, прізвище бригадира, азотні добрива, калійні добрива, фосфорні добрива.*

18. Розробити програму електронного довідника обліку врожайності пшениці. Робочі поля – *сорт пшениці, прізвище ланкової, номер поля, врожайність пшениці.*

19. Розробити програму електронного довідника обліку витрат посівного матеріалу. Робочі поля – *назва культури, прізвище ланкової, площа ділянки, витрати посівного матеріалу (ц).*

20. Розробити програму електронного довідника обліку виконання плану товарообігу за перше півріччя. Робочі поля – *назва місяця, план товарообігу, фактичне виконання товарообігу, відхилення.*

21. Розробити програму електронного довідника обліку тракторів. Робочі поля – *марка трактора, прізвище тракториста, рік випуску, вартість, розхід палива.*

22. Розробити програму електронного довідника обліку надоїв молока. Робочі поля – *номер ферми, прізвище доярки, кількість корів, валовий надій молока.*

23. Розробити програму електронного довідника обліку випуску цукру. Робочі поля – *назва цукрового заводу, прізвище директора, дата, вироблено цукру, реалізовано цукру.*

24. Розробити програму електронного довідника обліку оранки на зяб. Робочі поля – *дата, марка трактора, прізвище тракториста, площа зораної ділянки.*

25. Розробити програму електронного довідника обліку приплоду поросят. Робочі поля – *дата, прізвище свинарки, інвентарний номер свиноматки, кількість поросят, загальна жива вага поросят.*

26. Розробити програму електронного довідника аналізу посівного матеріалу. Робочі поля – *дата проведення аналізу насіння, назва культури, схожість насіння (%), чистота насіння (%), прізвище лаборанта.*

27. Розробити програму електронного довідника аналізу валового збору зернових культур. Робочі поля – *назва культури, прізвище бригадира, посівна площа, валовий збір зерна.*

28. Розробити програму електронного довідника аналізу плану заготовки кормів для ВРХ. Робочі поля – *вид корму, план заготовки, фактичне виконання плану на початок декади, фактичне виконання плану на кінець декади.*

29. Розробити програму електронного довідника обліку нарахування заробітної плати працівникам господарства. Робочі поля – *прізвище працівника, кількість відпрацьованих днів, відсоток виконання плану, заробітна плата.*

30. Розробити програму електронного довідника аналізу збору і реалізації урожаю культур. Робочі поля – *назва культури, валовий збір, держзамовлення, продано на ринку, залишилось у господарстві*

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Розробка граф-схем програми відповідно до завдання.
2. Розробка тексту програми на папері згідно з алгоритмом (програма має виводити обов’язково на екран свою назву та прізвище студента, що її розробив).
3. Запуск TURBO PASCAL.
4. Набір тексту розробленої програми та запис її у файл.
5. Налагодження програми згідно з завданням (знайти та виправити помилки), виконати розрахунки; результати показати викладачеві.

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЗАВДАННЯ

ПОСТАНОВКА ЗАДАЧІ

Задано квадратну матрицю A розміром N x N (N<=10), що складається з дійсних елементів. Знайти середнє арифметичне елементів кожного зі стовпців цієї матриці.

ТЕОРЕТИЧНЕ ВВЕДЕННЯ

Для обробки матриць у завданні застосовані вкладені оператори

ЦИКЛУ З ЛІЧИЛЬНИКОМ :

for <ідентифікатор>**:=**<нач. значення лічильника> ***to*** < кінцеве значення лічильника > ***do***

<оператор>

Для перевірки розмірності матриці (k<=N) застосований оператор

ЦИКЛУ ПОВТОРИТИ :

Repeat

<оператор>;

...

<оператор>;

Until <умова виходу з циклу>;

Введення фактичної кількості рядків і стовпців квадратної матриці **A** (**k<=N**), введення-висновок елементів матриці **A** і обчислення середнього арифметичного кожного зі стовпців матриці реалізовано через відповідні процедури: **Input, I.nputMatrix, OutputMatrix, Evaluate** з параметрами (див. Лістинг програми **Work4.pas**). Вихідні параметри передаються через атрибут **Var**.

Використовуються дві стандартні підпрограми модуля **CRT:**

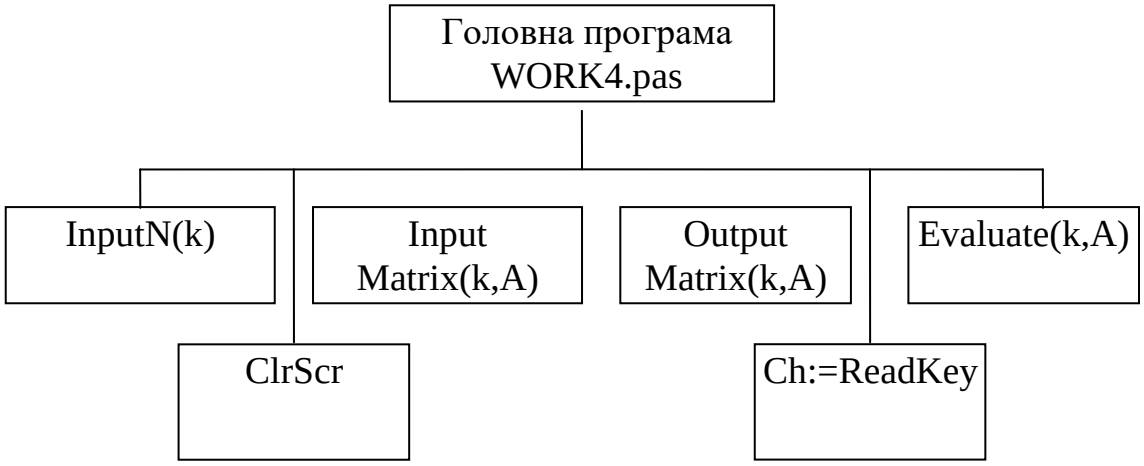
- Для очищення екрана – процедура **ClrScr.**
- Для введення символу (у даному випадку **n** чи **N**) – функція **ReadKey.**

ОПИС ПРОГРАМИ

Програма написана алгоритмічною мовою ПАСКАЛЬ і реалізована в середовищі **Borland Pascal-7.0 Windows**, процесор **Pentium**. Програма складається з головної програми і чотирьох підпрограм **(Input, InputMatrix, OutputMatrix, Evaluate)**, об’єднаних у єдиний модуль **WORK4.pas**. З головної програми викликаються зовнішні підпрограми

стандартного модуля **CRT**: **ClrScr, ReadKey.**

ОПИС ЛОГІЧНОЇ СТРУКТУРИ



ОПИС ВХІДНИХ ДАНИХ

k – фактична кількість рядків і стовпців матриці (**k<=N**) – перемінна типу **INTEGER**;

A – квадратна матриця, що складається з речовинних елементів (типу **REAL**):

A: array[1..N,1..N] of real.

ОПИС ВИХІДНИХ ДАНИХ

Stolb: array[1..N] of real – локальний масив середніх арифметичних значень елементів кожного зі стовпців матриці **A –** обчислюється і виводиться в процедурі **Evaluate.**

ОПИС ПІДПРОГРАМ

Процедура Input (Var k:integer)

Служить для уведення фактичної кількості рядків і стовпців квадратної матриці **A (1<k<=N**).

Процедура InputMatrix (k:integer; Var A:array)

Служить для введення значень речовинних елементів матриці **A** типу **Array (Array=array[1..N,1..N] of real)** довжиною **k*k.**

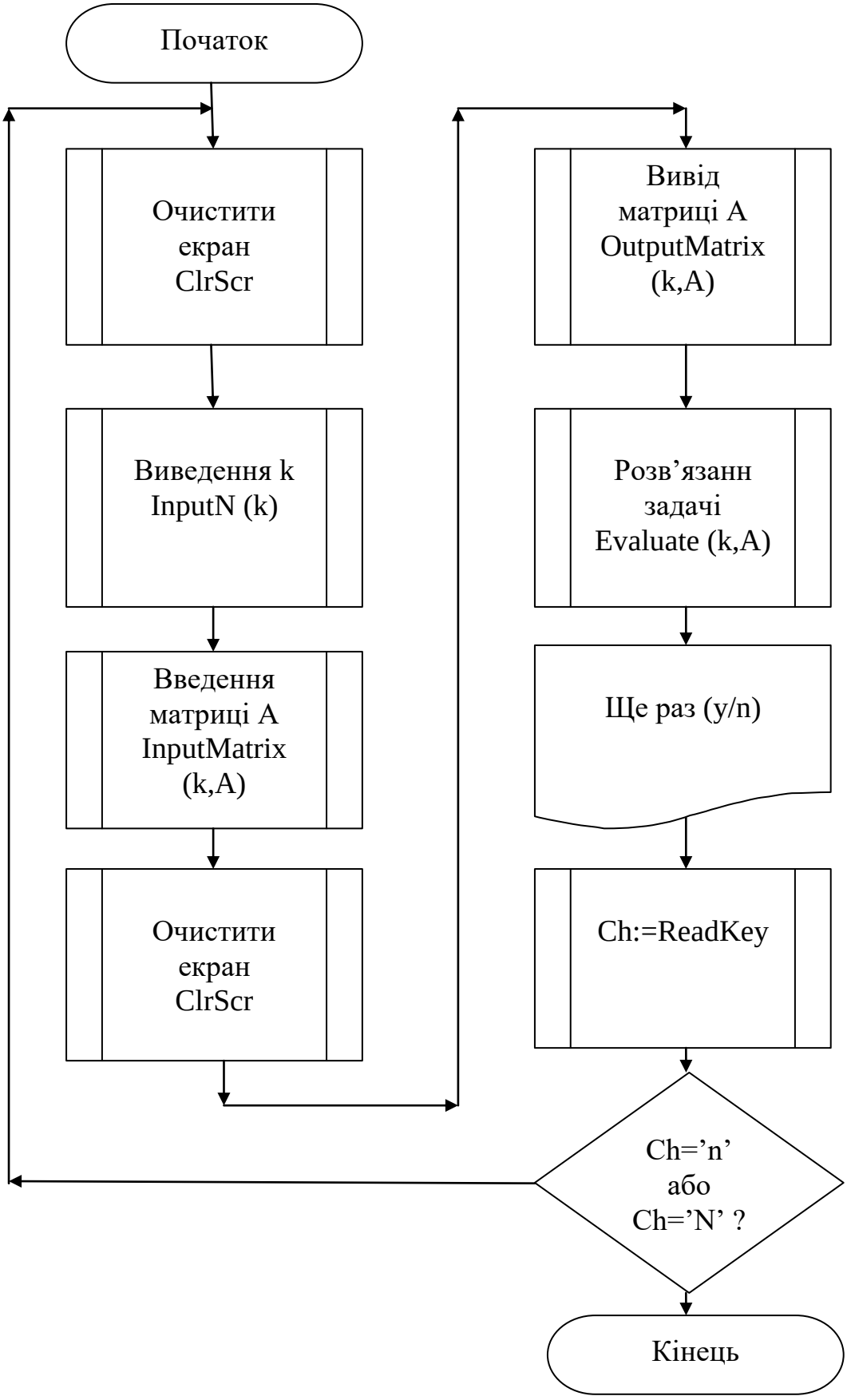
Процедура OutputMatrix (k:integer; A:array)

Служить для висновку значень речовинних елементів квадратної матриці **A** типу **Array** довжиною **k*k.**

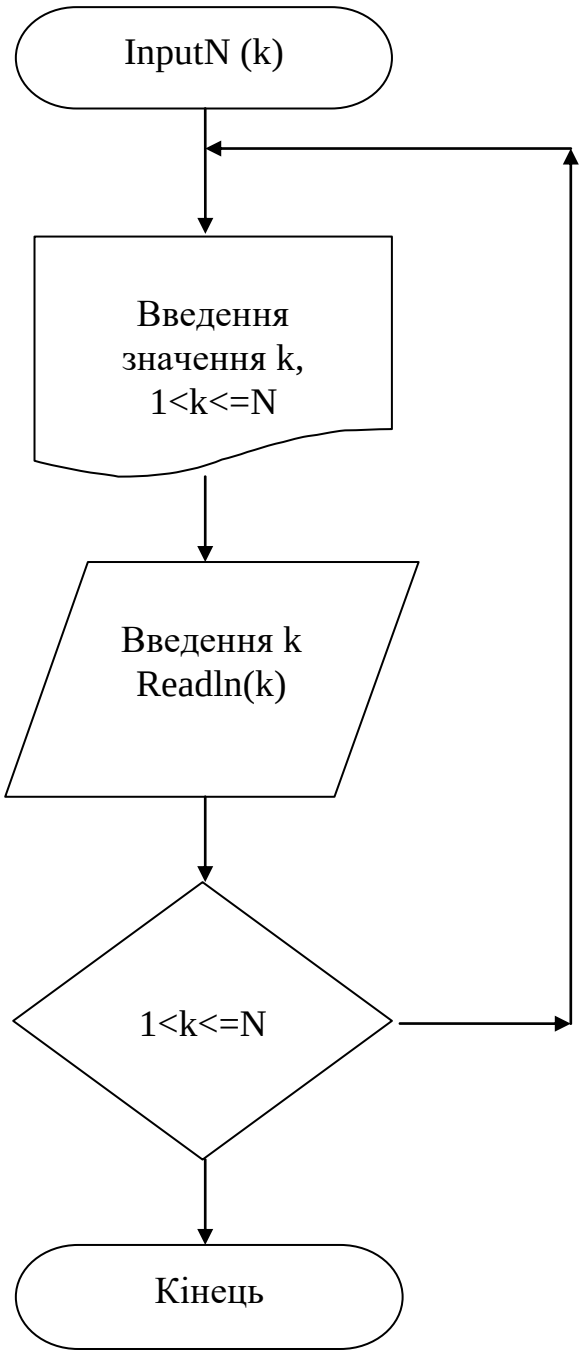
Процедура Evaluate (k:integer; A:array)

Служить для обчислення і видачі на екран середнього арифметичного кожного зі стовпців квадратної матриці **A** типу **Array** довжиною **k*k.**

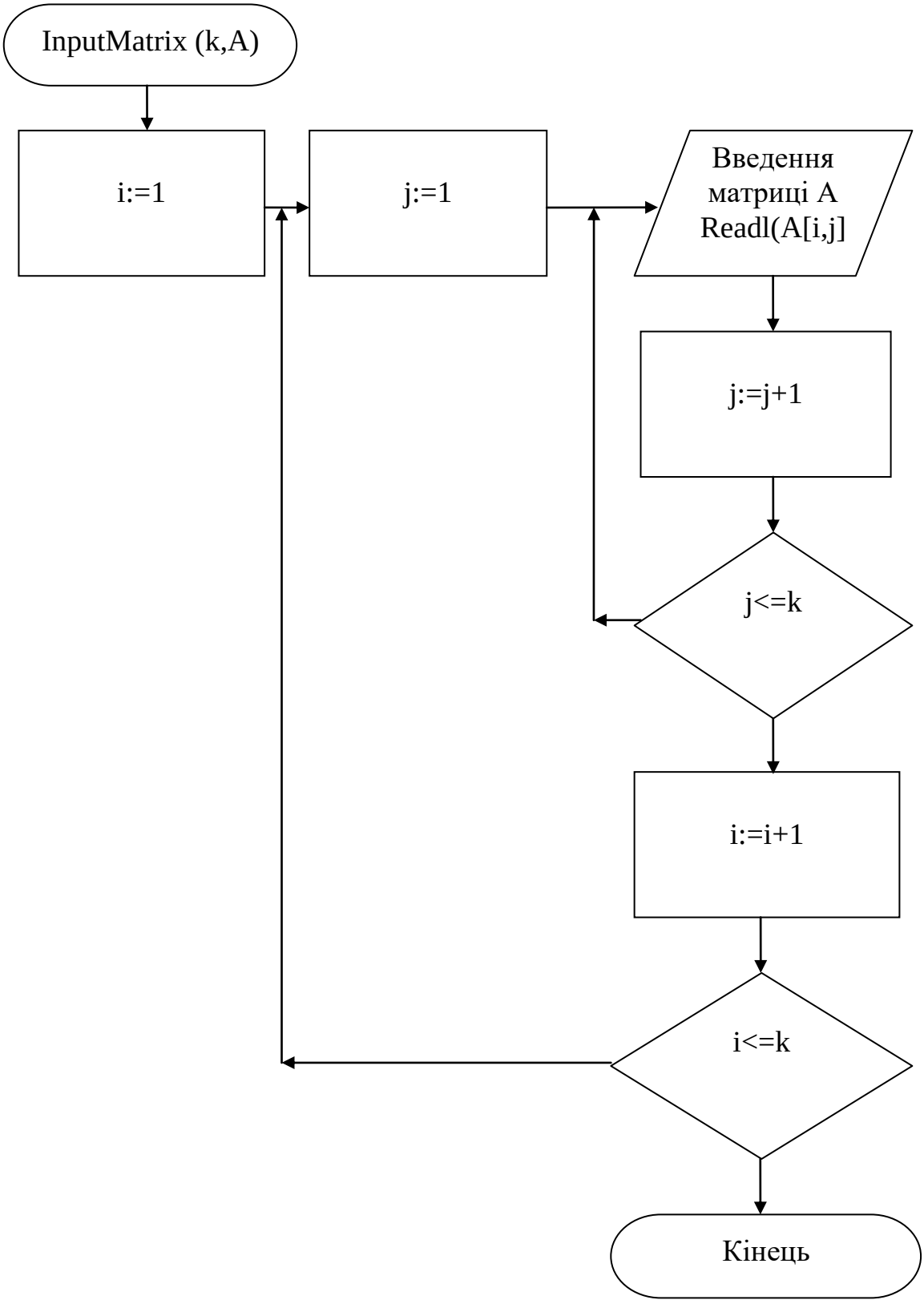
Головна програма WORK4.PAS



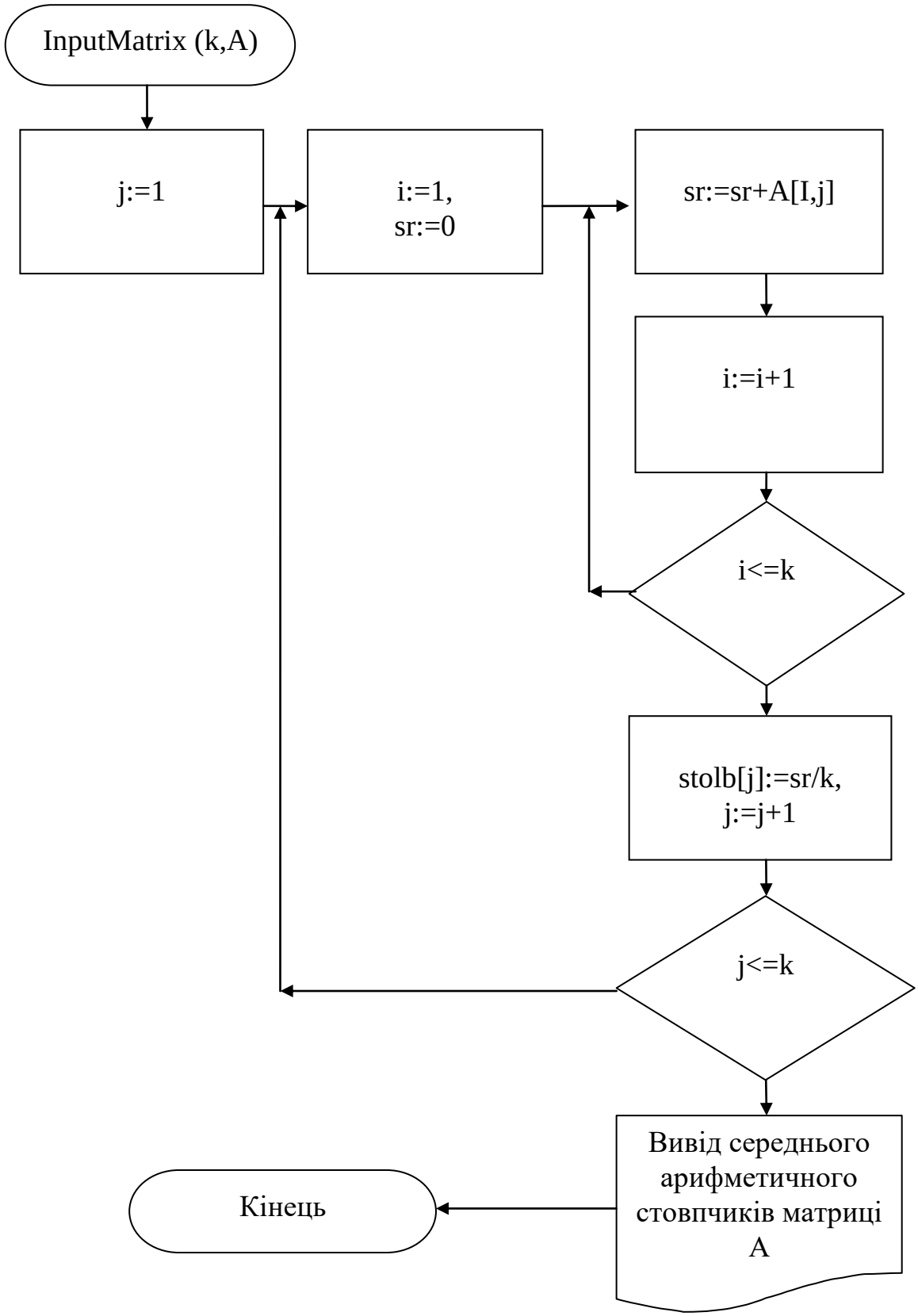
Процедура введення k – фактичної кількості рядків та стовпчиків квадратної матриці A



Процедура введення елементів матриці А



Процедура визначення середнього арифметичного кожного із стовпчиків матриці A



ТЕКСТ ПРОГРАМИ

```
Program Work4;
Uses CRT;
Const N=10;
Type ArrayA=array [1..N,1..N] of real;
Var k : integer;
  A: ArrayA;
  ch : char;

Procedure InputN(Var k:integer);
Begin
 Repeat
  Write('Введите значение N=====>');
  Readln(k);
 Until (k<=N) and (k>1);
End;

Procedure InputMatrix(k:integer;Var A:arrayA);
Var i,j:integer;
Begin
 for i:=1 to k do
 for j:=1 to k do
 Begin
   Write('Введите значение элемента матрицы A[',i,',',j,']=====>');
   Readln(A[i,j]);
 End;
End;

Procedure OutputMatrix(k:integer; A:arrayA);
Var i,j:integer;
Begin
 Writeln('   —— Исходная матрица ——');
 for i:=1 to k do
 Begin
   for j:=1 to k do
    Write(A[i,j]:7:2,' ');
   Writeln;
 End;
End;
```

```
Procedure Evaluate(k:integer; A:arrayA);
Var i,j:integer;
 stolb:array [1..N] of real;
 sr:real;
Begin
 for j:=1 to k do
 Begin
   sr:=0;
   for i:=1 to k do
    sr:=sr+A[i,j]; {вычисление суммы элементов по столбцам матрицы A}
   Stolb[j]:=sr/k;
 End;
 Writeln('—— Среднее арифметическое каждого из столбцов ——');
 for j:=1 to k do
  Writeln ('Столбец [',j,']= ',Stolb[j]:7:2);
End;

                    {Головна програма}

Begin
 Repeat
  ClrScr;
  InputN(k);
  InputMatrix(k,A);
  ClrScr;
  OutputMatrix(k,A);
  Evaluate(k,A);
  Writeln('Опять? (y/n)');
  ch:=ReadKey;
 Until (ch='n') or (ch='N');
End.
```

У **роботі** задані квадратна матриця **A розміром N * N (N<=10**), що складається з дійсних елементів. Необхідно знайти середнє арифметичне значення елементів кожного зі стовпців цієї матриці.

Ім'я програми – **WORK4.**

- У поле опису бібліотек і констант заданий стандартний модуль

CRT, визначена константа **N=10.**

- Описано власний тип даних **Array** для опису масиву розміром **N * N**, елементи якого мають тип **Real**, – **Type Array=Array[1..N,1..N]** of **Real.**
- Описано перемінні: цілочисельна **k: Integer,** масив **A:Array**; символьна перемінна **ch: char.**
- Далі йдуть **опису процедур Input, InputMatrix, OutputMatrix, і Evaluate.**
- Записано **головну програму**, що викликає потрібні процедури і функції, відповідно до алгоритму рішення задачі.

Опис процедури Input

- У заголовку процедури описаний один формальний параметр: вихідне **k** (значення визначається в процедурі і передається в основну про– граму).
- У циклі **Repeat ...Until (K<=N) and (K>1)** уводиться значення перемінної **K** – **Readln(K)** з перевіркою (тобто що вводиться **K** повинне бути більше **одиниці** і менше або дорівнює **N** – заданому за умовою максимальному значенню розміру масиву **A**). Потім керування передається в основну програму.

Опис процедури InputMatrix.

- У заголовку процедури описані параметри: **K –** вхідний параметр, переданий з основної програми, і **A** –вихідний параметр (масив елементів, що вводяться **в процедурі,** матриці А).
- Описано локальні цілочисельні перемінні **i** (лічильник числа рядків), **j** (лічильник числа стовпців) – **integer.**
- У процедурі в подвійному циклі **for** по перемінним **i, j** уводяться значення елементів масиву **A[I,J]: Readln(A[I,J].**
- Потім керування передається основній програмі.

Опис процедури OutputMatrix.

Оскільки алгоритм висновку матриці структурно нічим не відрізняється від алгоритму її введення і оформлення процедури майже однакове. Зверніть, будь ласка, увага на те, що масив **A** – тепер параметр ***вхідної (*** тобто ***відомий),*** тому ключове слово ***Var відсутнє***. У подвійному циклі **for** реалізований порядковий висновок на екран вихідної матриці. У форматі висновку під виведене значення елементів матриці приділяється 7 позицій, у тому числі дві позиції для висновку дробової частини:

Write(A[i,j]:7:2,' ').

Наступний оператор висновку без параметрів **Writeln** дозволяє закінчити висновок одного рядка матриці і перейти на наступну.

Опис процедури Evaluate

- У заголовку процедури описані параметри: **K, A** як вхідні параметри.
- Описано локальні перемінні **i** (лічильник числа рядків), **j** (лічильник числа стовпців) як **integer** і **stolb** (речовинний масив довжиною **N – array [1..N] of Real**), **s**r (робоча перемінна типу **Real** для підрахунку суми елементів у даному стовпці).
- У подвійному циклі **for** вважається середнє арифметичне значення для кожного стовпця і зберігається в одномірному масиві **Stolb:**
  1) цикл починається по ***стовпцях j від 1 до k*** ;
  2) потім ***у циклі по рядках i*** вважається сума елементів у даному стовпці **sr**;
  3) ***i*** обчислюється середнє арифметичне значення для кожного стовпця шляхом розподілу обчисленої суми на кількість рядків **Stolb[j]:=sr/k.**
- Значення елементів одномірного масиву середнього арифметичного значення стовпців **Stolb[j]** у циклі **for** виводиться на екран.

Зауваження. Середні арифметичні значення кожного зі стовпців матриці можна обчислювати і відразу видавати на екран. **Цей варіант придатний тільки тоді, коли значення середніх арифметичних більше ні для чого НЕ потрібні (як у нашому випадку).** Тоді потреба в масиві **Stolb** і додатковому циклі **for** відпадає. У цьому випадку підпрограма **Evaluate** буде мати наступний вид:

```
Procedure Evaluate(k:integer; A:arrayA);
Var i,j:integer;
  sr:real;
Begin
Writeln('–i–i–i– Середнє арифметичне кожного зі стовпців –i–i–i–');
for j:=1 to k do {–i–i–i– цикл по стовпцях –i–i–i– }
 Begin
   sr:=0;
   for i:=1 to k do {—— цикл по рядках —— }
            sr:=sr+A[i,j]; {обчислення суми елементів по стовпцях
            матриці A}
```

```
  Writeln (‘Стовпець [‘,j,’]= ‘, sr/k :7:2);
 End;
End.
```

Опис головної програми

- У тілі основної програми в циклі **Repeat ... Until** очищається екран (**ClrScr**).
- Шляхом виклику процедури **Input(k)** уводиться значення **k** – фактичне число рядків і стовпців вихідної матриці **A**.
- Шляхом виклику процедури **InputMatrix(k,A**) уводимо значення елементів вихідної матриці **A.**
- Знову очищаємо екран (**ClrScr**).
- Через виклик процедури **OutputMatrix(k,A**) виводимо на екран значення елементів вихідної матриці **A.**
- Через виклик процедури **Evaluate(k,A)** визначаємо значення середніх арифметичних кожного зі стовпців і виводимо їхній на екран.
- Потім випливає запит «**Знову? (Y/N**)».
- Перемінної **ch** привласнюється значення натиснутої клавіші **ch:=ReadKey** і в залежності від відповіді керування чи передається на початок циклу (**ch:=‘Y’**) для повторного рахунка, але вже з іншими вихідними даними, чи програма закінчує виконання (**ch:= ‘N’**).

ПИТАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

1. Комп’ютер як електронний автомат. Принцип виконання команд.
2. Узагальнена структура комп’ютера. Призначення його основних вузлів.
3. Електронні компоненти комп’ютерної техніки. Типи процесорів і комп’ютерів.
4. Електронні компоненти комп’ютерної техніки. Класифікація гнучких носіїв інформації та їх основні характеристики.
5. Електронні компоненти комп’ютерної техніки. Накопичувачі на жорстких магнітних дисках та їх основні параметри.
6. Носії інформації. Принцип розташування інформації на дискових носіях. Запис та зчитування інформації.
7. Порти комп’ютерів. Додаткові зовнішні пристрої (периферія).
8. Класифікація друкуючих пристроїв.
9. Структура файлів. Їх назва, розширення, дата тощо.
10. Принцип розташування файлів на магнітних носіях. Призначення секторів та службових зон (BOOT, FAT, ROOT).
11. Принцип читання файлів з диска.
12. Принцип запису файлів на диск.
13. Принцип виконання команд операційною системою. Програмні файли.
14. Поняття про алгоритмічні мови програмування.
15. Трансляція програм. Інтерпретатори, компілятори, відладчики.
16. Абетка, синтаксис та семантика TURBO-PASCAL.
17. Змінні та їх типи. Властивості змінних.
18. Оператор присвоювання.
19. Алгебраїчні вирази. Пріоритет операцій.
20. Оператор виведення інформації та правила його використання.
21. Оператор введення інформації та правила його використання.
22. Основні арифметичні функції TURBO-PASCAL та приклади їх використання.
23. Оператор безумовного переходу та мітки.
24. Оператори розгалуження програм та приклади їх застосування.
25. Селектор CASE та приклад його застосування.
26. Логічні операції та порядок їх виконання.
27. Оператор циклу FOR та принцип його роботи. Граф-схема циклів при збільшенні та зменшенні лічильника циклу.
28. Оператор циклу REPEAT-UNTIL та принцип його роботи. Граф-схема циклів при збільшенні та зменшенні лічильника цикла.
29. Оператор циклу WHILE-DO та принцип його роботи. Граф-схема циклів при збільшенні та зменшенні лічильника цикла.

30. Поняття про масиви та їх види. Індекси масивів. Правила присвоєння при роботі з масивами.
31. Рядкові та символьні змінні. Оператори роботи з рядковими змінними та принцип їх дії.
32. Текстовий режим екрану. Основні функції керування курсором, кольорами, виведенням та редагуванням інформації в текстовому режимі.
33. Текстовий режим екрану. Вікна та принципи роботи з ними.
34. Види графічних режимів та оператори що їх вмикають\вимикають. Система координат у графічному режимі.
35. Основні графічні оператори та принцип їх дії.
36. Оператори встановлення типу тексту та виведення його на екран в графічному режимі.
37. Поняття про текстовий файл. Оператори та принципи роботи з текстовими файлами.
38. Типізовані файли. Принцип зберігання інформації в таких файлах, особливість використання операторів Reset та Rewrite.
39. Оператори запису та читання інфомації з типізованих файлів.
40. Модульність програми. Принципи створення та виклику процедур.
41. Модульність програми. Принципи створення та виклику функцій.
42. Передача параметрів у процедуру та з неї. Глобальні параметри.
43. Передача параметрів у функцію та з неї.
44. В чому полягає метод послідовного пошуку екстремальних елементів при сортуванні масивів?
45. В чому полягає так званий «бульбашковий метод» сортування масивів?
46. Сортування обміном.
47. Бульбашкове сортування з просіванням.
48. Сортування вибором.
49. Сортування включеннями.
50. Бульбашкове сортування вставками.
51. Сортування методом знаходження мінімального елементу.
52. Сортування методом знаходження нового елементу.
53. Швидке сортування (сортування Хоара).
54. Сортування Шелла.
55. Шейкер-сортування.
56. Метод послідовного пошуку мінімумів.
57. Сортування деревом.
58. Пірамідальне сортування (турнірне сортування).
59. Сортування із злиттям.
60. Порозрядне цифрове сортування.
61. Модифікація бульбашкового сортування з ознакою.

62. Бульбашкове сортування із запам’ятовуванням місця останньої перестановки.
63. Модифікація простими вставками.
64. Сортування підрахунком.
65. Оболонкове сортування.
66. Сортування підрахунком розподілень.
67. Типізовані файли. Принцип зберігання інформації в таких файлах, особливість використання операторів Reset та Rewrite.
68. Оператори запису та читання інфомації з типізованих файлів.
69. Модульність програми. Принципи створення та виклику процедур.
70. Модульність програми. Принципи створення та виклику функцій.
71. Передача параметрів у процедуру та з неї. Глобальні параметри.
72. Передача параметрів у функцію та з неї.
73. Загальні положення теорії алгоритмів.
74. Визначення поняття алгоритму.
75. Інтуїтивне поняття алгоритму.
76. Основні вимоги до алгоритмів.
77. Визначення поняття алгоритму за Калмогоровим, Мальцевим та Марковим.
78. Машина Тьюрінга.
79. Операції над машинами Тьюрінга.
80. Рекурсивні функції.
81. Примітивно-рекурсивні функції.
82. Частково-рекурсивні функції.
83. Метод часткових цілей.
84. Метод підйому.
85. Метод відпрацювання назад.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Бардачов Ю.М., Соколова Н.А., Ходаков В.Є. Дискретна математика: підручник. К.: Вища шк., 2002. 287с.
2. Биков М.М. Дискретний аналіз і теорія автоматів : навч. посіб. Суми : Сум. держ. ун-т, 2016. 353 с.
3. Бразинська С. В., Дубовик Т. М. Дискретна математика для інформатиків : навч. посіб. Дніпро : ДВНЗ УДХТУ, 2018. 150 с.
4. Вербицький В. І., Колодяжний В. М., Лісіна О. Ю. Дискретна математика : навч. посіб. Харків : ХНАДУ, 2018. 183 с.
5. Григорків В. С., Ярошенко О. І. Дискретні моделі економічної динаміки : навч. посіб. Чернівці : Рута, 2014. 95 с.
6. Дейбук В. Г., Костенюк Н. Г., Вацек Д. О. Практичні заняття з дискретної математики : навч. посіб. Чернівці : Рута, 2019. 155 с.
7. Денисюк В.О. Дискретний аналіз. Методичні рекомендації для виконання лабораторних та самостійних робіт здобувачами вищої освіти галузі знань 05 Соціальні та поведінкові науки спеціальності 051 Економіка спеціалізації «Економічна кібернетика» першого (бакалаврського) освітнього ступеню. Вінниця: ВНАУ, 2017. 240 с.
8. Дивак М.П., Порплиця Н. П., Дивак Т. М. Ідентифікація дискретних моделей систем з розподіленими параметрами на основі аналізу інтервальних даних : монографія. Тернопіль : Екон. думка ТНЕУ, 2018. 219 с.
9. Дискретна математика : навч. посіб. / [Литвиненко О. Є. та ін.] ; Нац. авіац. ун-т. Київ : НАУ, 2017. 174 с.
10. Дискретна математика. Практикум : навч. посіб. / О. С. Манзій [та ін.] ; Нац. ун-т «Львів. Політехніка». Львів : Вид-во Львів. політехніки, 2016. 211 с.
11. Дискретні та алгоритмічні структури в інструментарії програмної інженерії : навч. посіб. / [В. В. Скалозуб та ін.]. Дніпропетровськ : Дніпропетр. нац. ун-т залізн. трансп. ім. В. Лазаряна, 2016. 254 с.
12. Коваленко Л.Б., Станішевський С.О. Дискретна математика для менеджерів : навч. посіб. для студентів ВНЗ. Харків : ХНУМГ ім. О.М. Бекетова, 2015. 280 с.
13. Кривий С. Л. Збірник задач з дискретної математики : навч. посіб для студентів ВНЗ. Київ ; Чернівці : Букрек, 2018. 455 с.
14. Медведєва М. О. Особистісно орієнтоване навчання дискретної математики засобами інформаційних технологій у вищих навчальних закладах : монографія. Умань : Жовтий О. О. [вид.], 2016. 233 с.
15. Правдюк Н.Л, Потапова Н.А., Волонтир Л.О. Економетрія: Навчальний посібник. 1-е вид.- Вінниця.: ПП Балюк І.Б., 2009.- 274 с.

16. Роїк М.В. Огляд програмних засобів статистичного аналізу даних/, М.В.Роїк, О.І.Присяжнюк, В.О.Денисюк // Ефективна економіка. – 2017. – № 7. – Режим доступу до журналу : http://www.economy.nayka.com.ua.

17. Трохимчук Р.М., Нікітченко М.С. Дискретна математика у прикладах і задачах : навч. посіб. Київ : Київський університет, 2017. 248 с.

18. Шинкаренко Л. М. Дискретна математика : навч. посіб. Кременчук : Щербатих О. В. [вид.], 2017. 293 с.

19. Юрчук Н.П. Використання економіко-математичних методів в управлінні інноваційним розвитком економічних систем/ Н.П. Юрчук//…

20. Юрчук Н.П. Економіко-математичні моделі прогнозування розвитку у галузі тваринництва/ Н.П. Юрчук//…

21. Юрчук Н.П. Теоретичні аспекти економетричного моделювання виробничої діяльності підприємств/ Н.П. Юрчук//…

Інформаційні ресурси

1. Блок схеми онлайн: як структурно представити інформацію? URL: http://chvv.com.ua/blok-shemi-onlajn-yak-strukturno-predstaviti-informatsiyu.

2. Інструменти для студентів-програмістів. URL: http://productivityblog.com.ua.

3. Корисні сервіси та сайти. URL: http://krnu.org/mod/page/view.php?id=3103.

4. Методичні матеріали з інформатики. URL: https://www.ua5.org/.

5. Онлайн середовище програмування PASCAL. URL: https://vasiliy-povkh.ucoz.ua/news/onlajn_seredovishhe_programuvannja_pascal.

6. Основи програмування на мові PASCAL. URL: http://pascal.dp.ua/.

7. Портал знань — Знання повинні бути доступними. URL: http://www.znannya.org/.

8. Codecademy. URL: https://www.codecademy.com.

9. Словник з інформатики. URL: http://xn--r1a3b.xn--b1amgblet.xn--j1amh/.

10. Тестування з мови Pascal. URL: http://programer.in.ua/index.php/testy/testy-z-movy-prohramuvannia-pascal.

11. Code. URL: https://code.org/.

12. Codeavangers. URL: https://codeavangers.com.

13. CodeCombat. URL: https://codecombat.com.

14. DOU: Спілка програмістів. URL: https://dou.ua/.

15. PASCAL. URL: https://lgs.lviv.ua/pascal/.

16. Pluralsight.URL: https://www.pluralsight.com/codeschool.

17. Programmr. Programmers Playground. URL: http://www.programmr.com.

18. Replace. Український форум програмістів. URL: https://Replace.org.ua.

19. Treehouse. URL: https://teamtreehouse.com.

20. W3schools. The world's largest web developer site. URL: https://www.w3schools.com.

АЛФАВІТНИЙ ПОКАЖЧИК

П

С

Ш

ДОДАТОК А

ПРАВИЛА ЗАПИСУ АРИФМЕТИЧНИХ ВИРАЗІВ МОВОЮ TURBO PASCAL

1. Два знаки арифметичної дії (+, –, *, /) підряд писати неприпустимо. Вони повинні розділятись хоча б круглими дужками.

2. Арифметичні дії виконуються у порядку пріоритетності: I – обчислення функцій; II – множення та ділення; Ш – додавання, віднімання. Операції однієї черги виконуються зліва направо.

3. Аргумент стандартних функцій береться у круглі дужки. Аргумент тригонометричних функцій представляється у радіанах.

4. Якщо в чисельнику є арифметичні операції «+» чи «–», то він береться у круглі дужки; якщо у знаменнику є будь-яка арифметична операція, то він береться у круглі дужки.

5. Послідовність виконання операцій визначається використанням круглих дужок: обчислення починаються із самих внутрішніх дужок. Число відкриваючих і закриваючих дужок у виразі повинно бути одинаковим.

6. При обчисленні виразу x^a останній замінюється виразом: $e^{aln(x)}$, де e – трансцендентне число, основа натурального логарифму (e = 2,71828182845904523536…), ($x>0$, $a>0$).

7. При обчисленні виразу a^x останній замінюється виразом: $e^{xln(a)}$, ($x>0$, $a>0$).

8. При обчисленні виразу $log_a(x)$ останній замінюється виразом: $ln(x)/ ln(a)$, де ln – натуральний логарифм ($x>0$, $a>0$).

9. Функція *tg(x)* при обчисленнях замінюється на еквівалентну, тобто:

$$\mathrm{tg}(x) = \sin(x) / \cos(x)$$

10. Наприклад, функції arccos(x) та arcsin(x) можливо представити через arctg(x):

$$\arccos(x) = arctg(\sqrt{1-x^2}\,/\,x);$$

$$\arcsin(x) = arctg(x/\sqrt{1-x^2}).$$

11. Значення числа π: π = 4*arctg(1), π = 3,14159265358979323846….

12. Значення кута α з градусної міри в радіанну: $\alpha_{рад} = \frac{\pi}{180} \cdot \alpha_{град}$

ДОДАТОК Б

ПОВІДОМЛЕННЯ ПРО ПОМИЛКИ TURBO PASCAL

Turbo Pascal генерує два типи повідомлень про помилки: повідомлення про помилки компіляції та повідомлення про помилки виконання.

Повідомлення про помилки компіляції

Якщо виникає помилка при компіляції, то Turbo Pascal або активізує вікно редагування і розміщує курсор на місці помилки в програмі, або видає повідомлення про помилку, рядок з помилкою та її номер; символ (^) в рядку вказує місцезнаходження помилки.

Номер помилки	Текст повідомлення	Коментар
1	Out of memory	Вихід за межі пам’яті
2	Identifier expected	Має бути ідентифікатор
3	Unknown identifier	Невідомий ідентифікатор
4	Duplicate identifier	Повторення ідентифікатора
5	Syntax error	Синтаксична помилка
6	Error in real constant	Помилка в константі дійсного типу
7	Error in integer constant	Помилка в константі цілого типу
8	String constant exceeds line	Рядкова константа не завершена
9	Too many nested files	Багато вкладених файлів
10	Unexpected end of file	Раптовий кінець файла
11	Line too long	Рядок дуже довгий
12	Type identifier expected	Має бути тип ідентифікатора
13	Too many open files	Дуже багато відкритих файлів
14	Invalid file name	Недопустиме ім’я файла
15	File not found	Файл не знайдено
16	Disk full	Диск переповнений
17	Invalid compiler directive	Недопустима директива
18	Too many files	Дуже багато файлів
19	Undefined type in pointer define	Невизначений тип в описі вказівника
20	Variable identifier expected	Має бути ідентифікатор
21	Error in type	Помилка в типі
22	Structure too large	Дуже довга структура
23	Set base type out of range	Кількість елементів у множині перевершує допустиме значення
24	File components can not be files or objects	Компоненти файла не можуть бути файлами чи об’єктами
25	Invalid string length	Недопустима довжина рядка
26	Type mismatch	Неспівпадання типів

Номер помилки	Текст повідомлення	Коментар
27	Invalid subrange base type	Недопустимий піддіапазон початкового типу
28	Lower bound than upper bound	Нижня межа більша за верхню
29	Ordinal type expected	Має бути тип, що перераховується
30	Integer constant expected	Має бути константа цілого типу
31	Constant expected	Має бути константа
32	Integer or real constant expected	Має бути константа цілого чи дійсного типу
33	Pointer Type identifier expected	Має бути ідентифікатор типу «вказівник»
34	Invalid function result type	Недопустимий тип результату
35	Label identifier expected	Має бути ідентифікатор мітки
36	BEGIN expected	Має бути оператор BEGIN
37	END expected	Має бути оператор END
38	Integer expression expected	Має бути вираз цілого типу
39	Ordinal expression expected	Має бути вираз типу, що перераховується
40	Boolean expression expected	Має бути вираз логічного типу
41	Operand types do not match	Невідповідність типів операторів
42	Error in expression	Помилка в виразі
43	Illegal assignment	Заборонене присвоєння
44	Field identifier expected	Має бути ідентифікатор поля
45	Object file too large	Об’єктний файл дуже великий
46	Undefined EXTERN	Невизначений зовнішній символ
47	Invalid object file record	Недопустимий запис об’єктного файла
48	Code segment too large	Сегмент команд дуже великий
49	Data segment too large	Сегмент даних дуже великий
50	DO expected	Має бути оператор DO
51	Invalid PUBLIC definition	Недопустиме визначення загального символу
52	Invalid EXTRN definition	Недопустиме визначення зовнішнього символу
53	Too many EXTRN definitions	Дуже багато визначень зовнішніх символів
54	OF–expected	Має бути оператор OF
55	INTERFACE expected	Має бути оператор INTERFACE
56	Invalid relocatable reference	Неприпустиме посилання на розташування в пам’яті
57	THEN expected	Має бути оператор THEN
58	TO or DOWNTO expected	Має бути оператор TO чи DOWNTO
59	Undefined forward	Невизначена функція чи процедура
60	Too many procedures	Дуже багато процедур
61	Invalid typecast	Несумісні типи змінних
62	Division by zero	Ділення на нуль
63	Invalid file type	Недопустимий тип файла

Номер помилки	Текст повідомлення	Коментар
64	Cannot read or write vars of this type	Неможливо писати чи читати змінні цього типу
65	Pointer variable expected	Має бути змінна типу «вказівник»
66	String variable expected	Має бути змінна рядкового типу
67	String expression expected	Має бути вираз рядкового типу
68	Circular unit reference	Циклічна залежність модулів
69	Unit name mismatch	Неспівпадання імені модуля
70	Unit version mismatch	Неспівпадання версії модуля
71	Duplicate unit name	Повторення імені модуля
72	Unit file format error	Помилка формату файла модуля
73	Implementation expected	Має бути оператор Implementation
74	Constant and case types don’t match	Неспівпадання типів константи і оператора CASE
75	Record variable expected	Має бути змінна типу «запис»
76	Constant out of range	Константа за допустимим діапазоном
77	File variable expected	Має бути змінна типу «file»
78	Pointer expression expected	Має бути вираз типу «вказівник»
79	Integer or real expression expected	Має бути вираз цілого чи дійсного типу
80	Label not within current block	Мітка за межами поточного блоку
81	Label already defined	Мітка вже визначена
82	Undefined label in preceding stmt part	Невизначена мітка в попередній частині оператора
83	Invalid @@argument	Недопустимий @@ аргумент
84	UNIT expected	Має бути оператор UNIT
85	«;» expected	Має бути символ «;»
86	«:» expected	Має бути символ «:»
87	«,» expected	Має бути символ «,»
88	«(« expected	Має бути символ «(«
89	«)» expected	Має бути символ «)»
90	«=« expected	Має бути символ «=«
91	«:=« expected	Має бути символ «:=«
92	«[« or «(« expected	Має бути символ «[« чи «(«
93	«]» or «)» expected	Має бути символ «]» чи «)»
94	«.» expected	Має бути символ «.»
95	«..» expected	Має бути символ «..»
96	Too many variables	Дуже багато змінних
97	Invalid FOR control variable	Недопустима змінна управління циклом
98	Integer variable expected	Має бути вираз цілого типу
99	Files and procedure types are not allowed here	Файли і типи процедур тут не дозволені
100	String length mismatch	Неспівпадання довжини рядка
101	Invalid ordering of fields	Недопустимий порядок полів
102	String constant expected	Має бути константа рядкового типу

Номер помилки	Текст повідомлення	Коментар
103	Integer or real variable expected	Має бути змінна цілого чи дійсного типу
104	Ordinal variable expected	Має бути змінна типу, що перераховується
105	INLINE error	Помилка в операторі INLINE
106	Character expression expected	Має бути вираз символьного типу
107	Too many relocation items	Дуже багато елементів у виразі
112	CASE constant out of range	Константа в операторі CASE за діапазоном
113	Error in statement	Помилка в операторі
114	Cannot call an interrupt procedure	Неможливо викликати процедуру обробки переривань
116	Must be in 8087 mode to compile	Повинен бути режим 8087 для компіляції
117	Target address not found	Необхідна адреса не знайдена
118	Include files are not allowed here	Неможливо підключити файл у цьому місці
120	NIL expected	Має бути оператор NIL
121	Invalid qualifier	Недопустиме використання
122	Invalid variable reference	Недопустиме посилання на змінну
123	Too many symbols	Дуже багато символів
124	Statement part too large	Дуже велика частина оператора
126	Files must be var parameters	Файли повинні бути параметрами змінної
127	Too many conditional symbols	Дуже багато символів в умовному виразі
128	Misplaced conditional directive	Пропущена умовна директива
129	ENDIF directive missing	Пропущена директива ENDIF
130	Error in initial conditional defines	Помилка у визначенні початкових умов виразів
131	Header does not match previous definition	Заголовки не відповідають по-передньому визначенню
132	Critical disk error	Критична помилка диска
133	Cannot evaluate this expression	Неможливо обчислити вказаний вираз
134	Expression incorrectly terminated	Вираз неправильно завершено
135	Invalid format specifier	Недопустимий формат специфікатора
136	Invalid inderect reference	Недопустимий непрямий вказівник
137	Structured variables are not allowed here	Структурні змінні не дозволені в цьому місці
138	Cannot evaluate without System unit	Неможливо обчислити без модуля System
139	Cannot access this symbol	Неможливо звернутись до вказаного символу
140	Invalid floating–point operation	Недопустима операція з дійсним числом

Номер помилки	Текст повідомлення	Коментар
141	Cannot compile overlays to memory	Неможлива компіляція оверлеїв у пам’яті
142	Procedure or function variable expected	Має бути процедура чи функція
143	Invalid procedure or function reference	Недопустимий вказівник процедури чи функції
144	Cannot overlay this unit	Неможливе використання цього модуля в оверлеї
145	Too many nested scopes	Дуже багато вкладень
146	File access denied	Звернення до файла заборонено
147	Object type expected	Має бути об’єктний тип
148	Local object types are not allowed	Локальний об’єктний тип заборонений (не дозволений)
149	Virtual expected	Має бути Virtual
150	Method identifier expected	Має бути ідентифікатор правила
151	Virtual constructors are not allowed	Віртуальні конструкції не дозволені
152	Constructor identifier expected	Має бути ідентифікатор конструктора
153	Destructor identifier expected	Має бути ідентифікатор деструктора
154	Fail only allowed within constructors	Відмова дозволена лише всередині конструктора
155	Invalid combination of opcode and operands	Недопустима комбінація коду операції та операндів
156	Memory reference expected	Має бути вказівник пам’яті
157	Cannot add or subtract relocatable symbols	Неможливе додавання чи віднімання символів, що переміщуються
158	Invalid register combination	Недопустима реєстрова комбінація
159	286/287 instructions are not enabled	Недоступні команди 286/287 процесорів
160	Invalid symbol reference	Недопустимий вказівник символу
161	Code generation error	Помилка генерації коду
162	ASM expected	Має бути ASM

Повідомлення про помилки виконання

Якщо при виконанні вашої програми виникає помилка, то процес виконання перерветься і з’явиться таке повідомлення:

Run–time error <nnn> at <xxxx:yyyy>,

де nnn – номер помилки виконання;

xxxx:yyyy – адрес помилки виконання.

Номер помилки	Текст повідомлення	Коментар
1	Invalid function number	Помилковий номер функції
2	File not found	Файл не знайдено
3	Path not found	Маршрут не знайдено

Номер помилки	Текст повідомлення	Коментар
4	Too many open files	Дуже багато відкритих файлів
5	File access denied	Звернення до файла заборонено
6	Invalid file handle	Недопустимий опис файла
12	Invalid file access code	Недопустимий доступ до файла
15	Invalid drive number	Недопустимий номер дисковода
16	Cannot remove current directory	Неможливо пересунути поточний каталог
17	Cannot rename across drives	Неможливо переіменувати дисководи
100	Disk read error	Помилка читання диска
101	Disk write error	Помилка запису на диск
102	File not assigned	Файл не призначено
103	File not open	Файл не відкрито
104	File not open for input	Файл не відкрито для введення
105	File not open for output	Файл не відкрито для виведення
106	Invalid numeric format	Недопустимий числовий формат
150	Disk is write–protected	Диск захищений від запису
151	Bad drive request structure length	Неправильна довжина структури запиту дисковода
152	Drive not ready	Дисковод не готовий
154	CRC error in data	Помилка контрольної суми даних
156	Disk seek error	Помилка при пошуку доріжки диска
157	Unknown media type	Невідомий тип носія
158	Sector not found	Сектор не знайдено
159	Printer out of paper	Принтер без паперу
160	Device write fault	Несправний пристрій запису
161	Device read fault	Несправний пристрій зчитування
162	Hardware failure	Збій апаратних засобів
200	Division by zero	Ділення на нуль
201	Range check error	Помилка контролю межі, діапазону
202	Stack owerflow error	Переповнення стеку
203	Heap owerflow error	Переповнення динамічної області
204	Invalid pointer operation	Недопустима операція з вказівником
205	Floating point owerflow	Переповнення розрядності дійсних чисел
206	Floating point underflow	Недопустима операція з дійсним числом
207	Invalid floating point operation	Недопустима операція з дійсним числом
208	Overlay manager not installed	Не встановлене управління оверлеями
209	Overlay file read error	Помилка читання оверлейного файла
210	Object not initialized	Об’єкт не ініціалізований
211	Call to abstract method	Виклик незрозумілого методу
212	Stream registration error	Помилка реєстрації потоку
213	Collection index out of range	Індекс колекції за межами
214	Collection overflow error	Помилка переповнення колекції

ДОДАТОК В
ГАРЯЧІ КЛАВІШІ ОБОЛОНКИ TURBO PASCAL

Клавіші	Функція
F1	Активізація вікна контекстно-залежної допомоги
Alt+F1	Повернення до попередньої довідки
Ctrl+F1	Активізація довідки про оператор, на який вказує маркер
Shift+F1	Виклик змісту довідкової підсистеми
F2	Збереження на диску файла з активного вікна
Ctrl+F2	Встановлення програмного лічильника на початок програми і закриття всіх раніше відкритих програмою файлів
F3	Відкриття нового вікна і загрузка в нього вибраного файла
Alt+F3	Закриття активного вікна
Ctrl+F3	Відкриття вікна протоколу використання процедур
F4	Виконання програми, розміщеної в активному вікні, до позиції курсора
Ctrl+F4	Перегляд і зміна значень змінних
F5	Зміна (збільшення/зменшення) розміру активного вікна
Alt+F5	Перехід на екран користувача
Ctrl+F5	Зміна положення і розміру вікна
F6	Перехід до наступного вікна
Shift+F6	Повернення до попереднього вікна
F7	Виконання пооператорно програми з пооператорним виконанням всіх підпрограм
Alt+F7	Перехід до попереднього рядка у вікні повідомлень
Ctrl+F7	Доповнення списку змінних у Watch–вікні
F8	Виконання пооператорно програми з виконанням підпрограм без пооператорної деталізації
Alt+F8	Перехід до наступного рядка у вікні повідомлень
Ctrl+F8	Встановлення/відміна контрольної точки на рядку програми, що вказується курсором
F9	Компіляція і редагування зв’язків програми
Alt+F9	Компіляція програми із активного вікна
Ctrl+F9	Компіляція і виконання програми
F10	Активізація рядка меню
Alt+F10	Виклик локального меню
Alt+Буква	Відкриття вибраною літерою підпорядкованого меню із рядка меню
Alt+Backspace	Відміна всіх змін в поточному рядку
Alt+X	Завершення сеансу роботи з Turbo Pascal із збереженням (після підтвердження) файлів, змінених редактором тексту
Alt+Цифра	Перехід до вікна з вказаним номером
Alt+0	Виклик вікна, в якому міститься список всіх відкритих вікон
Ctrl+Del	Видалення виділеного блоку
Ctrl+Ins	Копіювання блоку в буфер проміжного збереження
Shift+Del	Перенесення виділеного блоку в буфер проміжного збереження
Shift+Ins	Копіювання із буфера проміжного збереження у вікно редагування

ДОДАТОК Г

СТАНДАРТНІ МАТЕМАТИЧНІ ФУНКЦІЇ МОВИ TURBO PASCAL

Функція	Дія, що виконується
ABS(X)	Обчислює абсолютне значення (модуль) числа X
COS(X)	Обчислює косинус числа X, де X – кут у радіанах
SIN(X)	Обчислює синус числа X, де X – кут у радіанах
ARCTAN(X)	Обчислює арктангенс числа X. Результат – в радіанах
EXP(X)	Підносить e (основа натурального логарифма) до ступеня X
LN(X)	Обчислює натуральний логарифм від числа X
SQR(X)	Підносить число X до другого ступеня
SQRT(X)	Обчислює квадратний корінь із числа X
TRUNC(X)	Обчислює цілу частину числа X (дробова частина відкидається)
FRAC(X)	Обчислює дробову частину числа X
INT(X)	Обчислює цілу частину числа X
ROUND(X)	Число X заокруглюється до найближчого цілого числа
RANDOM(X)	Обчислюється випадкове число в інтервалі від 0 до X
ODD(X)	Результат буде TRUE, якщо число X непарне

ДЛЯ НОТАТОК

Навчальне видання

Коляденко Світлана Василівна

Денисюк Валерій Олександрович

Юрчук Наталія Петрівна

ДИСКРЕТНИЙ АНАЛІЗ

ЧАСТИНА І

Навчальний посібник

Набір і редагування авторські

Технічні редактори: *Денисюк Валерій Олександрович*
Юрчук Наталія Петрівна

Верстка

Підписано до друку Формат 60х84/16.
Папір офсетний. Ризографія. Авт. арк. 5,3.
Обл.-вид. арк. . Тираж 300 прим. Зам.____.

Підготовлено до друку та видруковано
у вищому навчальному закладі
«Вінницький національний аграрний університет».
Свідоцтво про внесення до Державного реєстру ДК № 1842.
21008, м. Вінниця, вул. Сонячна, 3.