Караванова Т. П.

ІНФОРМАТИКА

Основи алгоритмізації та програмування

10 клас

- Етапи розв'язування задач за допомогою комп'ютера
- Алгоритми і базові алгоритмічні структури
- Мова програмування Pascal
- Побудова алгоритмів
- Підпрограми

АСПЕКТ
aspekt-edu.kiev.ua

Караванова Т. П.

Інформатика

Базовий курс

Основи алгоритмізації та програмування (процедурне програмування)

Навчальний посібник

Видання четверте

Шепетівка
«Аспект»
2007

ББК 22.18
К-52

Технічні редактори, вчителі інформатики:
Ривкінд Й.Я. – ліцей № 38, м. Київ;

Рецензенти:
Тимофієва Є.М. – канд.фіз.-мат.наук, доцент Чернівецького національного університету імені Ю.Федьковича;
Чернікова Л.А. – старший викладач Запорізького обласного інституту післядипломної педагогічної освіти;

Коректор Філінюк Н.В.

Схвалено комісією з інформатики Науково-методичної ради з питань освіти Міністерства освіти і науки України (протокол № 1 від 12.01.2005 р.)

К-52 Інформатика. Базовий курс. Основи алгоритмізації та програмування/Караванова Т.П. – Шепетівка: «Аспект», 2007. – 192 с.

ISBN 978-966-2017-14-4

Матеріал, викладений у навчальному посібнику, відповідає вимогам «Програм для загальноосвітніх навчальних закладів фізико-математичного, природничого та технологічного профілів. Інформатика. 10-11 класи» (Програми для загальноосвітніх навчальних закладів. Інформатика. – Запоріжжя: Прем'єр, 2003. – 304 с.), рекомендованими МОН України.

Просто, доступно, лаконічно і досить повно представлені основи алгоритмізації, мови програмування Паскаль та описана робота в інтегрованому середовищі Turbo Pascal 7.0. Містить вправи і практичні роботи для закріплення набутих знань і навичок.

Рекомендується для учнів 10-11 класів загальноосвітніх навчальних закладів фізико-математичного, природничого та технологічного профілів. Відповідає вимогам діючих програм з інформатики та 12-бальній шкалі оцінювання знань учнів

ББК 22.18

ISBN 978-966-2017-14-4

Зміст

Вступ

Алгоритмізація, як розділ інформатики, вивчає методи і прийоми побудови алгоритмів та їх властивості. Метою даного посібника є, в першу чергу, розгляд цих питань та вироблення в учнів навичок побудови алгоритмів.

Зміст поняття «алгоритм» є ключовим у розділі алгоритмізації і його можна трактувати як зрозумілий і точний припис виконавцю щодо здійснення послідовності дій, спрямованих на досягнення поставленої мети. Це поняття є базовим і в теорії алгоритмів. Саме спроби формулювання самого загального поняття алгоритму призвели у 30-х роках XX сторіччя до виникнення теорії алгоритмів, яка вивчає питання побудови алгоритмічних моделей. Питання моделювання, видів моделей також будуть розглядатися в одній із тем цього посібника.

У посібнику мова йтиметься також і про практичний аспект, пов'язаний з поняттям алгоритму і з широким використанням цього поняття у програмуванні. Адже при необхідності реалізувати деякий алгоритм треба повідомити виконавцю програму дій. При цьому необхідно обрати спосіб опису алгоритму, який був би зрозумілим виконавцю. Якщо цим виконавцем є комп'ютер, то відповідно записом алгоритму повинна бути ***програма***, написана мовою «зрозумілою» комп'ютеру. Сам процес переведення записів алгоритмів на цю мову називається ***програмуванням***.

Отож, програмування можна розглядати як спосіб реалізації складених алгоритмів.

Для написання програми обирають мову програмування, а для того, щоб виконати її на комп'ютері, використовують відповідне середовище програмування.

Отож, надалі будемо говорити про мову Паскаль як про один із інструментів для реалізації алгоритмів, а про середовище програмування Турбо Паскаль 7.0 – один із засобів для отримання результату роботи цих алгоритмів.

Тепер можна підвести логічний підсумок наведених вище міркувань щодо визначення питання зв'язку між алгоритмом і програмою. Первинним є алгоритм, а програма, як один із варіантів реалізації алгоритму, є вторинною і використовується з метою отримання результату роботи алгоритму.

Інформаційна модель

1. Етапи розв'язування задач на комп'ютері

Перш ніж одержати очікуваний результат роботи програми на комп'ютері, необхідно виконати досить велику копітку підготовчу роботу. Усі ці дії можна сформулювати у такому вигляді:

- постановка задачі;
- побудова математичної моделі;
- побудова алгоритму;
- вибір мови програмування;
- складання програми;
- трансляція програми;
- налагодження програми, контрольний прорахунок;
- експлуатація програми.

Постановка задачі

Постановка задачі – це чітке формулювання умови задачі, визначення вхідних даних для її розв'язання, точні вказівки щодо визначення результатів, які мають бути отримані.

Розв'язування будь-якої задачі починається з її постановки. Подекуди безпосередньо в умові задачі присутні чітко визначені математичні поняття. На першому кроці необхідно добре уявити, в чому саме полягає дана задача, які необхідні для її розв'язання початкові дані, яку інформацію вважати результатами розв'язання цієї задачі і її майбутнє застосування.

Побудова математичної моделі

Побудова математичної моделі – це визначення сукупності математичних об'єктів і відношень між ними, які відображають деякі властивості процесу, що моделюється.

Побудова математичної моделі задачі – дуже відповідальний етап. Не завжди умова задачі містить в собі готову математичну формулу, яку можна застосувати для розробки алгоритму розв'язання задачі, не завжди розв'язок задачі вдається отримати в явному математизованому вигляді, що зв'язує вхідні дані та результати. Для цього створюється інформаційна математична модель об'єкта, що вивчається. Вибір виду моделі залежить від інформ-

маційної сутності об'єкта, а не від його фізичної природи. Тобто не на стільки важливе прикладне значення задач, скільки однотипність методів, якими вони розв'язуються. Приміром, логічні моделі використовуються як для моделювання людських міркувань, так і при описові логічних схем автоматики.

Наприклад, розглянемо задачу про рух тіла під дією прикладених до нього сил. Розв'язуючи її, ми, перш за все, записуємо рівняння руху тіла на основі класичних законів механіки. Але, крім сили тяжіння, на тіло діє і сила опору повітря. Постає питання відповідності математичної моделі реальній картині досліджуваного процесу. Подекуди буває неможливим врахування всіх реальних факторів, що впливають на нього. Тому дуже суттєвим є вміння виділяти серед всіх факторів головні та другорядні, для того, щоб останніми можна було знехтувати. При цьому може скластися ситуація, коли наперед невідомо, якими саме факторами можна знехтувати, і тому може бути декілька математичних моделей, що описують одну і ту ж саму задачу, явище з різним ступенем достовірності.

Ступінь відповідності моделі реальному об'єкту перевіряється практикою, комп'ютерним експериментом. Критерій застосування практики дає можливість оцінити побудовану модель і уточнити її в разі необхідності. Чим достовірніше математична модель відображає реальні сторони процесу, тим точніші результати, що отримуються.

Побудова алгоритму

Побудова алгоритму полягає у виборі того чи іншого способу розв'язування задачі.

Наступним кроком є розробка алгоритму обробки інформації на основі математичної моделі. Саме про створення алгоритмів, як послідовності виконання певних вказівок для досягнення мети, йтиметься мова у цьому посібнику. Для побудови алгоритму необхідно визначити спосіб розв'язування поставленої задачі. Можуть бути застосовані вже відомі методи, проведена їх оцінка, аналіз, відбір, або розроблені нові методи. Наприклад, вибір методу розв'язування системи рівнянь, що описує розроблену математичну модель.

Вибір мови програмування

Оптимальний вибір мови програмування базується на реальному оцінюванні складності та характеру задачі.

Алгоритм, призначений для виконання на комп'ютері, повинен бути записаний певною мовою програмування. Різноманітність

існуючих мов програмування вимагає від програміста реальної оцінки складності та характеру задачі. Тільки після цього він зможе здійснити вибір найоптимальнішої мови програмування для реалізації поставленої задачі. Враховуючи можливість розбиття всього алгоритму на окремі частини, для кожного з них вибір мови програмування може бути здійснений окремо.

Складання програми

Складання програми полягає у створенні тексту програми на основі розроблених алгоритмів і представлень інформації з урахуванням алфавіту та правил запису обраною мовою програмування.

Цей етап не потребує додаткового пояснення, а лише вимагає знання тієї мови програмування, що була обрана. Суть його полягає в тому, щоб на основі розроблених алгоритмів і представлень інформації створити програму для комп'ютера.

Трансляція програми

Трансляція програми – це переклад програми на машинну мову.

Переклад програми на машинну мову здійснюється за допомогою спеціальних програм – трансляторів. Однією з їх функцій є перевірка наявності у програмі синтаксичних помилок після введення її у комп'ютер. Не тіште себе надією, що програма, навіть найпростіша, написана бездоганно. Серед програмістів відоме таке прислів'я: «Якщо у програмі немає помилок, це означає, що у вас поганий транслятор»!!!

Налагодження програми, контрольний прорахунок

Налагодження програми, контрольний прорахунок – це перевірка правильності роботи програми за допомогою системи тестів.

Виправлення усіх синтаксичних помилок у програмі на попередньому етапі зовсім не позбавляє від іншого типу помилок – змістовних, логічних. Вони з'являються при помилковому трактуванні умови поставленої задачі, недосконалості математичної моделі або недоліках в алгоритмі та призводять до отримання помилкового результату. Такі помилки не можуть бути усунені на стадії трансляції, тому що для їх виявлення необхідна інформація про сутність самої задачі. Змістовні (логічні) помилки може усунути лише сам розробник.

Смисл налагодження програми полягає в тому, що готується система тестів, за допомогою якої перевіряється робота програми у різних можливих режимах. Кожний тест містить набір вхідних даних, для яких відомий результат. Для більшості програм виникає необхідність добору не одного, а цілої серії тестів, щоб перевірити найбільше число можливих ситуацій, які можуть статися в процесі роботи програми. Якщо для всіх вдало підібраних тестів ваші розрахунки співпали з результатами роботи програми, то можна вважати, що логічних помилок немає.

Експлуатація програми

Експлуатація програми – це використання програми замовниками або користувачами.

Якщо ваша програма успішно пройшла усі попередні етапи, то можна з певною мірою припущення вірити всім результатам, одержаним при будь-яких початкових даних. Тепер вашу програму можна тиражувати і пропонувати іншим користувачам, доповнивши тексти програм описом обмежень на початкові дані, відповідною документацією для користувача, описом середовища даної програми тощо.

Більшість розглянутих етапів розв'язання задач на комп'ютері виконуються людиною і носять творчий, евристичний характер.

Питання для самоконтролю:

1 *Які існують етапи розв'язування задачі на комп'ютері?*
2 *Що означає вираз «постановка задачі»? Хто її виконує?*
3 *Що означає вираз «побудова математичної моделі»?*
4 *Що означає вираз «побудова алгоритму»?*
5 *Що означає вираз «вибір мови програмування»?*
6 *Як перевіряється правильність алгоритму?*
7 *Які помилки можуть бути в алгоритмі?*
8 *Які помилки називаються синтаксичними?*
9 *Які помилки називаються логічними? Як їх визначити і виправити?*
10 *Як отримати машинний код програми?*
11 *Як здійснюється налагодження і тестування програми?*
12 *До якого етапу можна переходити після налагодження програми?*

2. Інформаційна модель

Поняття інформаційної моделі

Розглянувши етапи розв'язування задач на комп'ютері, ми торкнулися одного виду моделювання – створення математичної моделі. Однак слід розширити поняття моделі, як засобу опису досліджуваного об'єкта.

Під моделлю деякого об'єкта розуміється сукупність найважливіших властивостей об'єкта, процесу чи явища, яка володіє суттєвими для цілей моделювання властивостями і в рамках цих цілей повністю замінює вихідний об'єкт, процес чи явище.

Модель – це не тільки зовнішня подібність, головне лежить глибше, оскільки поведінка моделі і реального об'єкта повинні підкорятися однаковим закономірностям. Вивчивши їх на доступній для дослідження моделі, можна передбачити властивості досліджуваного об'єкта.

Галузь знань, що займається розробкою різноманітних моделей, їх теорією і використанням, називається моделюванням.

Існує велика кількість моделей, які відрізняються складністю, різноманітністю задач та цілей моделювання, галузями застосування. Моделювання базується на двох методах дослідження:

- експериментальному, де використовуються *предметні моделі;*
- теоретичному, при якому використовуються різного роду знакові, *абстрактні моделі.*

До ***предметних*** або матеріальних моделей можна віднести моделі зовнішньої подібності.

Серед них можна виділити *експериментальні моделі* – манекени, іграшки, моделі літаків, кораблів, на яких проводять попередні випробування.

Функціональні моделі заміняють об'єкти при виконанні певних функцій: пристрої штучної нирки, систему «серце-легені», протези, маніпулятори тощо.

Тренажери, моделі, що імітують поведінку реальних об'єктів в складних ситуаціях, слугують для навчання і відносяться до *навчальних моделей.*

Ті моделі, які не тільки статично відображають основні властивості об'єкта, але ще й у часі імітують їх носять назву імітаційних. Імітаційне моделювання пов'язане з моделюванням динамічних об'єктів, процесів, явищ. Зміна ситуацій в часі – той фактор, який вивчається за допомогою імітаційних моделей. Наприклад, диспетчер на залізниці, що дивиться на табло, по якому в умовному вигляді переміщуються потяги, має справу з відображуваною моделлю. Але якщо табло знаходиться в навчальному центрі, де навчають майбутніх диспетчерів, то процеси, що відображаються на табло, лише імітують реальність. В цьому випадку іде мова про імітаційне моделювання.

До абстрактних або знакових моделей слід віднести в першу чергу математичні моделі, які за допомогою математичних формул описують досліджуваний об'єкт. Існують і інші знакові моделі, які базуються на науковому апараті конкретної галузі науки. У зв'язку з цим розрізняють, окрім математичних, ще й фізичні, біологічні, соціологічні моделі тощо. Зразком цих абстрактних моделей є ілюстрації у шкільних підручниках з кожного предмету, формули, схеми, таблиці тощо.

Абстрактні знакові моделі – це такі, які можна зобразити у вигляді набору знаків. Дитячий малюнок, який зображає схід сонця, літній день, відпочинок на морі, конструкторська схема літака, інформація в електронній таблиці та результати її обробки тощо – все це абстрактні моделі.

Серед абстрактних моделей слід виділити ті, які ще не існують на папері, комп'ютері або на якомусь іншому носії інформації. Перш ніж така модель буде зафіксована на певному носієві, вона з'являється у людини у вигляді ідеї, вона може обговорюватись, тобто передаватись усно від однієї людини до іншої. Такі абстрактні моделі, які ще не є знаковими, називаються вербальними, тобто усними.

Але для реалізації будь-якої моделі на комп'ютері переважно застосовується математичний апарат. Побудові математичної моделі передує постановка задачі, на основі якої формулюється інформаційна модель. Вважається, що інформатика як наука починається тоді, коли для фрагменту світу, що вивчається, побудована інформаційна модель. Отож, математичні моделі, на яких базується комп'ютерне моделювання, тісно пов'язані з інформаційними моделями.

Інформаційна модель – це інформація про об'єкт чи процес, яка описує деякі важливі для конкретної розв'язуваної задачі його типові риси та властивості.

У якості найпростіших прикладів інформаційних моделей можна навести такі:

- в рецептурі виготовлення гастрономічної страви описується, якими повинні бути овочі та фрукти – зрілі або напівзрілі, відварені, смажені або сирі, та яким подається до столу готове блюдо – гарячим, охолодженим, як гарнір або як основна страва;

• опис дитячої гри містить умови її проведення: де і як повинна відбуватися гра (на вулиці, у приміщенні, за столом тощо), скільки гравців може брати у ній участь, хто вважається в решті решт переможцем;

• для виготовлення нової сукні треба знати якість тканини, з якої її можна виготовити, метраж тканини, розміри сукні;

• зміст довідкових видань та енциклопедій;

• умови виконання домашнього завдання з математики – наявність підручника, зошита, ручки, олівця, лінійки, та його результат – записи розв'язків задач у зошиті з урахуванням правил їх виконання.

Мал.1

Інформаційні моделі вже давно застосовуються у фізиці, хімії, механіці та інших галузях науки. Останніми роками знайшло широке розповсюдження інформаційне трактування біологічних процесів, зокрема вищої нервової діяльності. Інформаційна модель, відображаючи найбільш суттєві властивості реального досліджуваного об'єкта, насправді нетотожна йому, а є лише наближеним його описом.

Такі моделі це ті ж відносні істини, через посередництво яких пізнається реальна дійсність з постійним наближенням до істини.

Інформаційні моделі є тим містком, через який інформатика вступає у відношення з окремими науками, не зливаючись з ними і в той же час не вбираючи їх у себе.

Хоча загальні методологічні принципи побудови інформаційних моделей можуть бути предметом дослідження інформатики, сама побудова і обґрунтування інформаційної моделі є задачею окремої науки.

На завершення систематизуємо інформацію про моделі у вигляді схеми (мал.1):

Побудова інформаційної моделі

Практично всі дослідження оточуючого нас світу дають нам неповну, наближену інформацію, але це не заважає людям здійснювати польоти у космос, пізнавати таємниці атомного ядра, оволодівати законами суспільного розвитку тощо. Необхідно, щоб побудована наближена модель найбільш повно відображала властивості об'єкту, явища чи процесу, що вивчаються. Наближений характер моделі може проявлятися по різному. Наприклад, отримані результати, що використовуються в експерименті, на які впливає точність показання приладів, розклад польотів літаків, складений без урахування метеоумов.

Правильно побудувати модель – один із найважливіших етапів дослідження об'єкта, явища чи процесу. Для цього необхідно визначити основні його властивості, виснити, вникаючи в сутність поставленої задачі, що задано і які результати необхідно отримати. Мабуть, найголовніше – це оцінити множину вхідних даних. Критеріїв відбору вхідних даних є два. Перший – це наявність та відповідний ступінь залежності досліджуваного результату від цих вхідних даних. Другий – це можливість практичного включення даного фактору в модель, що будується.

Оскільки модель будується на деякому спрощеному описові об'єкта, результати, отримані при аналізі моделі, носять наближений характер. Ступінь відповідності моделі та об'єкта визначає і ступінь точності отриманих результатів. Ця відповідність перевіряється практикою, експериментом. Критерій практики дає можливість оцінити побудовану інформаційну модель та за необхідності уточнити її.

Розглянемо декілька прикладів побудови інформаційної моделі, яка передує подальшій побудові математичної моделі та розробці алгоритму розв'язування поставленої задачі.

Розглянемо реалізацію задачі про пошиття шкільної форми для учнів початкової школи. Оскільки мається на увазі пошиття не індивідуального, а типового одягу, то при розробці лекал необхідно знехтувати особливостями фігури кожної окремої дитини, взявши до уваги типові розміри дітей цього віку. Окрім цього якість, ціна та колір тканини визначаються однаковими для всіх форм даної моделі. Тому при розробці моделі шкільної форми для учнів початкової школи необхідно взяти до уваги типові лекала розмірів дітей віку від 6 до 10 років, тканину для пошиття костюмів, кольорів синього, коричневого або бордового.

Розробка алгоритму прогнозу погоди передбачає для побудови інформаційної моделі визначення тих параметрів, які будуть визначати хід аналізу і дослідження цього явища. Цими параметрами повинні обов'язково бути напрямок переміщення теплих і холодних фронтів, зміна тиску повітря, розташування і вплив на погоду в регіоні циклонів та антициклонів тощо. Однак можна знехтувати такими параметрами як вплив землетрусів, техногенних катастроф у віддалених районах земної кулі, масове паління осіннього листя на відкритому повітрі тощо.

Побудова інформаційної моделі визначення коренів квадратного рівняння передбачає забезпечення умов виконання цієї задачі. Такими умовами є визначення коефіцієнтів цього рівняння. Коефіцієнтами можуть бути будь-яких три числа за однієї умови: число, що є коефіцієнтом при x2, не повинно дорівнювати нулю.

Питання для самоконтролю:

1 Що називається моделлю?
2 Що називається моделюванням?
3 На яких двох методах дослідження базується моделювання?
4 На які типи можна поділити всі моделі?
5 Наведіть приклади предметних моделей.
6 Які моделі відносяться до абстрактних?
7 Наведіть схему класифікації моделей.
8 Що таке інформаційна модель?
9 Яке місце в процесі розв'язування задачі займає інформаційна модель?
10 Поясніть принципи побудови інформаційної моделі.
11 Наведіть власні приклади побудови інформаційної моделі.
12 Як визначається відповідність моделі та досліджуваного об'єкта?

Алгоритми

3. Поняття алгоритму

Поняття алгоритму

Алгоритмом називається скінчена послідовність вказівок, виконання яких призводить до досягнення поставленої мети.

У далекому IX-му столітті жив та працював відомий середньоазіатський мудрець, вчений, філософ, математик Мухаммед ал-Хорезмі. У своїх наукових трактатах він детально пояснив правила виконання арифметичних дій. При перекладі цих наукових робіт на латину вперше з'явився термін "алгоритм» (ал-Хорезмі – Algorithmi) і використовувався він спершу для визначення правил обчислень у десятковій позиційній системі числення.

Сучасне поняття слова «алгоритм» більш широке. Воно для багатьох співзвучне зі словами метод, спосіб, процедура, програма. Можна сказати, що алгоритм – це чітко сформульована інструкція, а інструкції зустрічаються практично у всіх сферах нашого життя.

Алгоритм є фундаментальним поняттям інформатики. Науковці виділяють три основних класи алгоритмів: обчислювальні, інформаційні та керуючі.

Обчислювальні алгоритми – це такі, в ході виконання яких проводяться обчислення. Обчислювальні алгоритми працюють з числами або з їх наборами – векторами, матрицями, множинами.

Інформаційні алгоритми працюють з великими об'ємами інформації. Як приклад можна навести алгоритм пошуку необхідної числової або символьної інформації, що відповідає певним ознакам. Ефективність роботи цих алгоритмів залежить від організації даних, прикладом чого можуть бути відомі сьогодні всім бази даних.

Керуючі алгоритми характерні тим, що дані до них надходять від зовнішніх процесів, якими вони керують. Результатами роботи цих алгоритмів є вироблення своєчасного необхідного керуючого сигналу як реакції на швидку зміну вхідних даних. Саме тому значення даних у ході виконання керуючих алгоритмів змінюються, подекуди навіть досить швидко, і алгоритм повинен своєчасно правильно прореагувати на це, тобто видати потрібний керуючий сигнал у потрібний момент часу.

У 30-х роках XX століття виникла теорія алгоритмів. До цього часу поняття алгоритму зводилось до набору елементарних кроків:

арифметичних дій, перевірки рівностей, нерівностей та інших відношень такого типу. Але на початку XX століття об'єкти, з якими оперували алгоритми, почали ускладнюватися, з'явилась необхідність виконувати операції над векторами, таблицями, множинами тощо. Постали питання щодо трактування поняття елементарності кроків, тлумачення однозначності алгоритму, виникла думка, що не для всяких математичних задач можна знайти алгоритм розв'язання за кінцевий проміжок часу. Теорія алгоритмів досліджує питання побудови конкретних алгоритмічних моделей, кожна з яких містить конкретний набір елементарних кроків, способів визначення наступного кроку. Завданням теорії алгоритмів є також дослідження питання про існування чи не існування ефективних алгоритмів розв'язання окремих задач. Найбільшу цінність представляють моделі, які одночасно були б і універсальними, і простими.

Бурхливий розвиток обчислювальної техніки, використання її в дослідженнях багатьох наук привів до створення великої кількості різноманітних алгоритмів в різних прикладних галузях. Зрозуміло, що всякому алгоритму відповідає задача, яку він призначений розв'язувати, але разом з тим може існувати не один алгоритм, що розв'язує дану задачу. Такі алгоритми називаються еквівалентними, і, зрозуміло, постає питання вивчення ефективності цих алгоритмів. Дослідники цих питань створили новий розділ теорії алгоритмів – теорію складності алгоритмів.

Складність алгоритму – це кількісна характеристика. Вона визначається часом, за який виконується алгоритм (часова складність), та об'ємом пам'яті комп'ютера, необхідного для його виконання (ємкісна складність). Тому про складність алгоритму логічно вести мову стосовно саме машинних алгоритмічних моделей. Оскільки подолання обмежень на пам'ять комп'ютера – технічна проблема, що вирішується на рівні його вдосконалення майже що півроку, то часова складність алгоритму, яка в більшій мірі залежить від обраної алгоритмічної моделі, методу реалізації задачі – творча, евристична проблема.

На практиці користувачів більше цікавлять не самі алгоритми, а задачі, які можна розв'язувати за їх допомогою. А оскільки для розв'язування задачі існують різні алгоритми, тому природно серед всіх визначити той, який має найменшу складність. Саме такі питання в межах даного посібника будуть цікавити нас найбільше:

- знайомство із засобами створення алгоритмів;

- знайомство з алгоритмами для розв'язування типових задач;
- порівняння алгоритмів, що розв'язують однакові задачі.

Приклади алгоритмів

Як зазначалося вище, виникнення поняття алгоритмів пов'язане з математикою. І саме у шкільному курсі математики ви часто зустрічалися з алгоритмами. На уроках з арифметики вивчалися алгоритми покрокового додавання, множення, ділення, добування квадратного кореня, пізніше на уроках з математики вас знайомили з правилами побудови бісектриси кута, ділення відрізка навпіл та на задану кількість рівних частин за допомогою олівця, циркуля та лінійки. Не обійшлося без правил та законів у таких науках, як фізика та хімія.

Наприклад, вам відомі алгоритми проведення фізичних та хімічних експериментів, дослідження різних явищ. На уроках з гуманітарних предметів ви вивчали безліч різних правил правопису.

З алгоритмами ви зустрічаєтеся і у повсякденному житті. Наприклад, у розпорядку робочого дня учня день від дня мало чим відрізняється: прокинутися вранці, вмитися, вдягнутися, поснідати, піти до школи, відвідати уроки, повернутися додому, пообідати, зробити домашнє завдання, відпочити до вечора, повечеряти, лягти спати. І так щодня.

В техніці добре відомі алгоритми обробки деталей, в побуті – складання меблів, користування електричними приладами тощо.

Отже, мало того, що ми живемо у суцільному інформаційному просторі, та ще й виявляється навколо нас багато алгоритмів, які усе життя нам доводиться виконувати!

Властивості алгоритму

Визначивши основне поняття алгоритму, можна зробити висновок, що будь-який алгоритм є послідовністю деяких вказівок. Але не кожна така послідовність може називатися алгоритмом.

Отже, сформулюємо основні властивості алгоритму.

Дискретність – будь-який алгоритм зображується у вигляді окремих вказівок.

Виконання команд алгоритму повинно бути послідовним, з точною фіксацією моментів завершення виконання однієї команди і початку виконання наступної.

Скінченність – виконання алгоритму завершується після завершення кінцевої кількості кроків.

В математиці існують обчислювальні процедури, які мають алгоритмічний характер, але не володіють властивістю скінченності. Прикладом такої процедури може бути обчислення значення числа π. Така процедура описує нескінченний процес і ніколи не завершиться. Але якщо обмежитися певною кількістю знаків після коми, то ми отримаємо алгоритм обчислення числа π з заданою точністю.

Визначеність – кожний крок алгоритму повинен бути чітко і недвозначно визначений, не повинен допускати довільного трактування виконавцем.

Тобто алгоритм розрахований на механічне виконання. Якщо один і той самий алгоритм доручити для виконання різним виконавцям, то вони повинні дійти одного і того ж результату.

Зрозумілість – формулювання дій алгоритму повинно бути орієнтоване на конкретного виконавця.

Якщо виконавець є некомпетентною людиною в питаннях, що вирішуються даним алгоритмом, то необхідно вибрати доступнішу для його формулювання форму, якою є словесний спосіб. Наприклад, якщо алгоритм передбачає обчислення коренів деякого квадратного рівняння, то для учня початкових класів необхідно самим детальним чином описати всі виконувані дії на рівні, що відповідає його знанням математики: додавання, віднімання, множення, ділення, добування квадратного кореня за допомогою калькулятора тощо. Для учня старших класів цей алгоритм буде містити математичну формулу для обчислення коренів рівняння та аналіз їх існування та відсутності.

Якщо ж виконання алгоритму буде запропоновано комп'ютеру, то алгоритм потрібно зобразити насамкінець відповідною машинною мовою.

Масовість – в алгоритмі повинна бути передбачена можливість виконання його для різних початкових значень.

Цим самим забезпечується його використання для розв'язування цілого класу однотипних задач.

Результативність – алгоритм повинен забезпечувати обов'язкове отримання результату після кінцевої кількості кроків.

Тобто кожна дія повинна бути достатньо простою, щоб її можна було виконати точно і за скінчений проміжок часу.

Як бачимо, для того, щоб набір інструкцій можна було назвати алгоритмом, він повинен відповідати ряду умов.

Виконавець алгоритму

Визначення поняття виконавця є досить неоднозначним. В деяких випадках виконавцем алгоритмів є людина. Наприклад, якщо мова йде про правила, інструкції, функціональні та посадові обов'язки тощо. Але людина – далеко не єдиний виконавець алгоритмів. Роботи-маніпулятори та верстати з програмним управлінням, жива клітина і навіть тварини в цирку виконують різноманітні алгоритми, в тому числі і ті алгоритми, які людина не в змозі виконати.

Уявимо роботу виконавця, як пристрою керування. Він «розуміє», тобто вміє виконувати, послідовність вказівок (алгоритми) і організує їх виконання, при цьому керуючи відповідними інструментами. А інструменти виконують дії, реалізуючи при цьому команди керуючого пристрою. Якщо людину розглядати як виконавця, то можна провести таку аналогію: мозок – пристрій керування, руки, ноги, очі, ніс і т. і. – інструменти. У роботів-маніпуляторів, верстатів з програмним управлінням, комп'ютерів пристроєм керування є процесор, а набір інструментів залежить від того, для розв'язання яких задач призначений той чи інший виконавець.

Важливо зрозуміти основні характеристики виконавця алгоритму: середовище, в якому повинен виконуватися алгоритм; елементарні дії, виконання яких потребує алгоритм; система команд алгоритму; відмови, які можуть відбуватися під час виконання алгоритму.

Середовище виконання алгоритму для виконавця-людини залежить від призначення та мети даного алгоритму. Можна говорити про установу, як середовище, де необхідно виконувати відповідні функціональні та посадові обов'язки; місце відпочинку, де треба дотримуватись правил поведінки на дорозі, на воді, у місцях загального користування; школа, де учні повинні дотримуватись правил поведінки на уроці, перерві, виконувати домашні завдання з різних предметів тощо. Для виконавця-комп'ютера середовищем виконання алгоритму є відповідне програмне забезпечення. Якщо користувач набирає текст, тобто виконує певну послідовність вказівок щодо підготовки тексту, то виконавцем цього алгоритму є текстовий редактор. Якщо користувач складає програму, то виконавцем алгоритмів набирання тексту програми, налагодження

її, запуску на виконання буде середовище програмування, що відповідає обраній мові програмування.

Кожний виконавець може виконувати команди лише із деякого строго заданого списку – системи команд виконавця. Наприклад, при приготуванні кожної страви визначений спосіб підготовки та застосування окремих компонентів, що входять до рецептури: нарізання, смаження, охолодження, змішування тощо. При виконанні домашніх завдань для учня визначені такі команди: читання, запам'ятовування, повторення, запис домашніх завдань у зошит, виконання завдань на комп'ютері і т.і. Для реалізації алгоритмів деякою мовою програмування необхідно знати правила запису алгоритмів цією мовою програмування та визначені в ній оператори.

У свою чергу виклик команди спонукає виконавця здійснити відповідну елементарну дію. Наприклад, з наведеного прикладу приготування страви для виконання команди нарізання необхідно виконати такі елементарні дії: приготувати дошку для нарізання, взяти ніж, продукти, що нарізуються, тощо. При записові учнем домашнього завдання необхідно взяти зошит, ручку, прочитати завдання, записати відповіді у зошит. При виконані на комп'ютері команди додавання значень двох величин спочатку визначаються адреси розміщення цих величин у пам'яті комп'ютера, вичитуються ці значення, сумуються, визначається адреса розміщення у пам'яті результату виконання команди та записується за цією адресою отриманий результат.

Для кожної команди повинні також бути задані умови застосування, тобто чітко визначено, в яких станах середовища може бути виконана команда, і описані результати виконання команди. Наприклад, пральна машинка-автомат буде працювати лише за наявності водопостачання. За його відсутності режим прання не увімкнеться. Найвищий бал учню буде виставлено лише за умови творчого підходу до виконання ним сформульованого завдання.

Слід зазначити, що при виконанні команд можуть виникати відмови їх виконання виконавцем. Ці відмови виникають під час виклику команди в неприпустимому для даної команди стані середовища. Ситуації, при яких виникає відмова, різні для різних команд виконавця. Наприклад, при обчисленні значення дробу можливе ділення на нуль. Але існують команди, при виконанні яких відмови ніколи не виникають. Наприклад, обчислення комп'ютером або калькулятором виразу «2+2» завжди є коректним.

Під формальним виконанням алгоритму розуміється таке його виконання, коли сам виконавець не знає ні постановки задачі, ні змісту одержаних результатів, але, чітко виконуючи усі дії, записані в алгоритмі, досягає необхідного результату.

Найчастіше алгоритми виконуються саме формальним чином. Формальними виконавцями є користувачі різноманітних побутових електричних приладів, учасники спортивних ігор, які дотримуються правил цих ігор, учні – при використанні у своїх творчих роботах мовних правил, при виконанні завдань з математики та фізики, використовуючи відомі формули, закони та методи розв'язання різних типів задач. Продовжуючи цю думку, слід вважати комп'ютер також формальним виконавцем алгоритмів при виконанні ним програм. Адже на сьогоднішній день більшість алгоритмів розрахована саме на виконання комп'ютером.

Аргументи та результати алгоритму

Для виконання багатьох алгоритмів необхідно задавати початкові значення. Ці значення передаються в алгоритм за допомогою аргументів.

Аргументи – це величини, значення яких необхідно задати для виконання алгоритму.

Правда, деколи зустрічаються алгоритми, що не вимагають ніяких початкових значень для свого виконання. Пізніше буде нагода познайомитися з такими алгоритмами. Однак немає жодного алгоритму, що не дає ніякого результату. Дійсно, який же смисл у такому алгоритмові? Прикладом різноманітності результатів роботи програм є ігрові комп'ютерні програми. Одержувана ними під час роботи закодована інформація певним чином перетворюється у графічні та звукові образи.

Результати – це величини, значення яких одержуються внаслідок виконання алгоритму.

При складанні багатьох алгоритмів виникає необхідність окрім аргументів та результатів використовувати ще додаткові величини. Введення в алгоритм таких величин залежить від самого автора алгоритму.

Проміжні величини – це величини, які додатково вводяться в ході розробки алгоритму.

Ми вже достатньо наочно уявили собі алгоритм. Але як же він виглядає з точки зору звичайного користувача? Частіше за все користувач не знає, яким чином цей алгоритм записаний, за

допомогою яких команд, які методи були застосовані для його реалізації, якою мовою програмування. Звичайно, що швидше за все в даному випадку мається на увазі виконання алгоритму на комп'ютері у вигляді готової програми. Виконавець алгоритму бачить лише його зовнішню сторону: які початкові дані необхідно задати і у якому вигляді одержується результат. Кожний з вас, напевно, може пригадати номер фокусника, коли той у чорну скриньку закладає одні предмети, а витягає зовсім інші. Від зору глядачів прихований вміст цієї чорної скриньки і залишається таємницею, яким чином в ній відбувається «перетворення» предметів. Саме такою «чорною скринькою» для користувача є і програма, якою він користується на комп'ютері.

Це можна зобразити таким чином:

Питання для самоконтролю:

1 *Що ми називаємо алгоритмом? Як виник термін «алгоритм»?*
2 *Наведіть власний приклад алгоритму.*
3 *Назвіть основні властивості алгоритмів.*
4 *Що означає «дискретність алгоритму»? Наведіть приклад.*
5 *Що означає «скінченність алгоритму»? Наведіть приклад.*
6 *Що означає «визначеність алгоритму»? Наведіть приклад.*
7 *Що означає «зрозумілість алгоритму»? Наведіть приклад.*
8 *Що означає «масовість алгоритму»? Наведіть приклад.*
9 *Що означає «результативність алгоритму»? Наведіть приклад.*
10 *Що називається середовищем виконання алгоритму?*
11 *Що називається системою команд виконавця?*
12 *Наведіть приклади систем команд виконавця.*
13 *Що називають відмовами виконавця, що можуть траплятися під час виконання ним алгоритму?*
14 *Що розуміється під формальним виконанням алгоритму?*
15 *Що ми називаємо аргументами алгоритму?*
16 *Що розуміється під результатами алгоритму?*
17 *Коли виникає необхідність введення проміжних даних?*

4. Способи запису алгоритмів

Ми познайомилися з поняттям алгоритму. Тепер нам необхідно домовитися, яким чином ми будемо його записувати.

Існує чотири способи запису алгоритмів, вибір яких залежить від того, хто його складає або на кого він орієнтований:

- словесний спосіб запису алгоритмів;
- запис алгоритмів за допомогою схем;

- описування алгоритмів мовою псевдокодів;
- запис алгоритмів мовою програмування.

Словесний спосіб запису алгоритмів орієнтований на людину-виконавця.

Мабуть, поки що важко уявити собі інший спосіб запису. Це найбільш проста і доступна форма представлення алгоритму. Словесна форма зазвичай використовується для алгоритмів, орієнтованих на виконавця – людину і цей спосіб є найбільш доступним будь-кому незалежно від його спеціальної підготовки.

Повертаючись до історії трансформації поняття алгоритм, слід зазначити, що ще до 1950 року під цим словом частіше за все розуміли викладений в «Елементах» Евкліда алгоритм Евкліда – процес знаходження найбільшого спільного дільника (НСД) двох натуральних чисел. Тому буде корисно навести для прикладу опис цього алгоритму.

- Взяти два натуральних числа. Якщо вони рівні, то перше з них і є найбільшим спільним дільником, якщо ж ні, то перейти до пункту 2.
- Порівняти два числа і вибрати більше з них.
- Більше з двох чисел замінити різницею більшого і меншого.
- Перейти до пункту 1.

Зверніть увагу, що у першому пункті конкретно сказано, яке саме число необхідно вибрати у випадку співпадання двох чисел. Саме це і є характерною рисою алгоритму, виконавцем якого може бути навіть не підготовлена в даній галузі людина.

Наведений алгоритм застосуємо до конкретної пари чисел.

Нехай задані два числа: 45 та 12.

Продемонструємо процес знаходження НСД за наведеним алгоритмом у вигляді таблички, де на кожному кроці більше число, від якого віднімається друге (менше) число, виділятимемо жирним шрифтом.

1) **45** 12	5) **9** 3
2) **33** 12	6) **6** 3
3) **21** 12	7) 3 3
4) 9 **12**	

Отже, за сім кроків ми одержали результат: **НСД(45,12)=3.**

Перевірка роботи алгоритму є суттєвим кроком на шляху до його розуміння. Надалі слід домовитися, що кожний алгоритм, наведе-

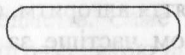
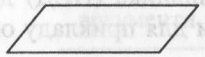
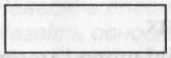
ний в посібнику, повинен бути розібраний по кроках. Це найпростіший і найефективніший спосіб розуміння будь-якого алгоритму.

Схеми дозволяють зобразити алгоритм в наочній графічній формі.

Цей спосіб вже вимагає деяких знань. Вони полягають у знайомстві зі спеціальними стандартами графічних зображень блоків, в середину яких поміщаються команди алгоритму.

Наведемо таблицю деяких з цих блоків.

Таблиця 1

	Початок та кінець алгоритму
	Введення або виведення даних
	Вибір напрямку виконання алгоритму в залежності від виконання умови
	Виконання операцій, в результаті яких відбувається зміна значення даних

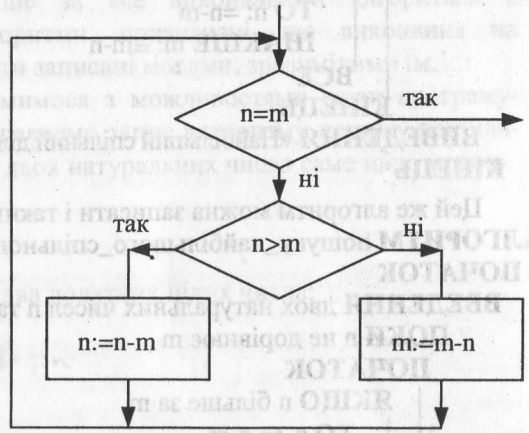
При створенні схеми алгоритму блоки із записаними в них командами з'єднуються між собою стрілками, які визначають черговість виконання дій алгоритму.

Для запису команд всередині блоків використовується природна мова з елементами математичної символіки. В результаті перевірки умови під час вибору напрямку виконання алгоритму виникають два можливі шляхи для його продовження. Ці шляхи зображуються стрілками з позначеннями «так» (+) та «ні» (–). Перехід по стрілці з позначенням «так» відбувається у випадку, коли умова виконується, а перехід по стрілці з позначенням «ні» – у протилежному випадку.

Блоки початку і кінця алгоритму використовуються при записові повного алгоритму задачі. Ми ж надалі вважатимемо, що алгоритми, які розглядаються в посібнику як базові, можуть бути самі використані в інших алгоритмах як їх фрагменти. Тому блоки початку і кінця алгоритму в таких випадках нами використовуватись не будуть. Для запису алгоритмів, з якими ми матимемо справу, даної інформації цілком достатньо. Зобразимо у визначених позначеннях алгоритм Евкліда.

Погодьтеся, що наочність схематичного представлення алгоритму (мал.2) має свої переваги!

Однак ця наочність швидко втрачається, коли зображується великий алгоритм. В таких випадках в схемі алгоритму виділяються і відокремлюються її окремі частини – модулі, основною умовою яких є один вхід і один вихід. Згодом вони включаються у схему алгоритму як окремі блоки. Такий підхід до складання алгоритму відображає ідею структурного програмування, про яке ще не один раз буде далі йти мова.



Мал.2

Для запису алгоритмів за допомогою мови псевдокодів використовуються службові слова та спеціальні правила запису окремих дій.

У мові псевдокодів прийняті певні синтаксичні правила для запису команд, що полегшує запис алгоритму на стадії його проектування і дає можливість використання більш широкого набору команд, розрахованого на абстрактного виконавця.

Наприклад, у мові псевдокодів використовуються спеціальні службові слова та правила запису для вибору напрямку подальшого виконання алгоритму в залежності від виконання чи невиконання сформульованої умови **ЯКЩО...ТО...ІНАКШЕ**, повторення визначеної групи дій певну кількість разів **ПОКИ...ПОЧАТОК...КІНЕЦЬ** тощо. Разом з тим мова псевдокодів дозволяє використовувати дещо довільну форму представлення математичних записів, умов тощо. Розглянемо як приклад алгоритм Евкліда:

АЛГОРИТМ найбільший_спільний_дільник
ПОЧАТОК
ВВЕДЕННЯ «Задайте два натуральних числа», n,m

```
ПОКИ n≠m
  ПОЧАТОК
  | ЯКЩО n>m
  | | ТО n: =n-m
  | | ІНАКШЕ m: =m-n
  | ВСЕ
  КІНЕЦЬ
ВИВЕДЕННЯ «Найбільший спільний дільник заданих чисел: », n
КІНЕЦЬ
```

Цей же алгоритм можна записати і таким чином:

```
АЛГОРИТМ пошуку_найбільшого_спільного_дільника
ПОЧАТОК
  ВВЕДЕННЯ двох натуральних чисел n та m
    ПОКИ n не дорівнює m
      ПОЧАТОК
      | ЯКЩО n більше за m
      | | ТО n ⇐ n-m
      | | ІНАКШЕ m ⇐ m-n
      | ВСЕ
      КІНЕЦЬ
  ВИВЕДЕННЯ значення найбільшого спільного дільника заданих чисел n
  КІНЕЦЬ
```

У наведеному алгоритмі зовсім неважко розібратися. Мова псевдокодів максимально наближена до звичайної розмовної мови. Усі службові слова, що використовуються для запису алгоритму, виділені жирним шрифтом, їх необхідно записувати тільки таким чином, як це передбачено правилами мови. Окремі блоки алгоритму виділені вертикальними лініями і означають певні дії. Внутрішній блок визначає вибір тієї чи іншої дії в залежності від відношення між значеннями величин n та m. Середній блок визначає дію повторення внутрішніх дій до того часу, поки значення величин n та m не стануть рівними. Зовнішній блок визначає повністю весь алгоритм, в якому передбачені також дії введення та виведення відповідно вхідної та результуючої інформації.

Як видно із наведених прикладів мову псевдокодів можна вважати проміжною між природною і формальною мовою. Для запису алгоритмів мовою псевдокодів необхідно знати всі службові слова цієї мови та правила запису деяких дій алгоритму.

*Найбільш високий **професійний рівень** для запису алгоритмів визначається знанням мов програмування.*

На практиці частіше за все виконавцями алгоритмів є комп'ютери. Тому алгоритми, призначені для виконання на комп'ютерах, повинні бути записані мовами, зрозумілими їм.

Надалі ми познайомимося з можливостями мови програмування Паскаль, тому розглянемо запис алгоритму пошуку найбільшого спільного дільника двох натуральних чисел саме цією мовою.

```
program NSD;
 var n,m:    integer;
begin
   writeln ('Задайте два додатних цілих числа:');
   readln (n,m);
   while n<>m do begin
   if n>m
     then n:=n-m
     else m:=m-n;  end;
   writeln ('Найбільший спільний дільник заданих чисел:',n)
end.
```

Ви звернули увагу на подібність запису алгоритму мовою псевдокодів та мовою програмування Паскаль? Це дійсно так, але для запису алгоритму на Паскалі потрібно орієнтуватися в англійській мові та вміти використовувати усі можливості цього потужного засобу програмування.

Питання для самоконтролю:

1 *Які ви знаєте способи запису алгоритмів?*
2 *Які ви знаєте правила словесного запису алгоритму?*
3 *Чому словесний спосіб запису алгоритму вважається найпростішим?*
4 *В чому особливості схематичної форми зображення алгоритму?*
5 *У чому полягають особливості мови псевдокодів?*
6 *Обґрунтуйте твердження, що мови програмування – це один із способів подання алгоритму.*
7 *Чим обумовлена така послідовність способів опису алгоритмів: словесний, схема алгоритму, мова псевдокодів, мови програмування?*

5. Базові структури алгоритмів. Типи алгоритмів

Після ознайомлення з різними формами запису алгоритмів зможемо розпізнати різні за типами команди, а відповідно і алгоритми, які їх використовують. Скористаємося для цього знайомим алгоритмом Евкліда.

Будь-який алгоритм можна уявити собі як деяку структуру, що складається з окремих базових елементів. Зрозуміло, що при такому підході до алгоритмів вивчення основних принципів їх конструювання слід починати з ознайомлення з цими базовими елементами.

Визначають три *базових структурних елементи*: лінійний, розгалужений, циклічний.

Лінійним елементом алгоритму називається така операція, яка визначає один елементарний крок обробки або відображення інформації.

До таких операцій відносяться дії зміни значення деяких величин, введення та виведення інформації тощо (мал.3).

Мал.3

Декілька лінійних елементів можуть об'єднуватися і утворювати складену лінійну структуру або лінійний фрагмент алгоритму. В алгоритмі Евкліда лінійними елементами є дії введення та виведення інформації, обчислення нових значень величин *n* та *m*.

У якості прикладів лінійних елементів можна навести такі:

- натиснення кнопки POWER при вмиканні комп'ютера;
- перехід вулиці, по якій заборонено рух транспорту;
- обчислення дискримінанту квадратного рівняння;
- виведення результату обчислення арифметичного виразу.

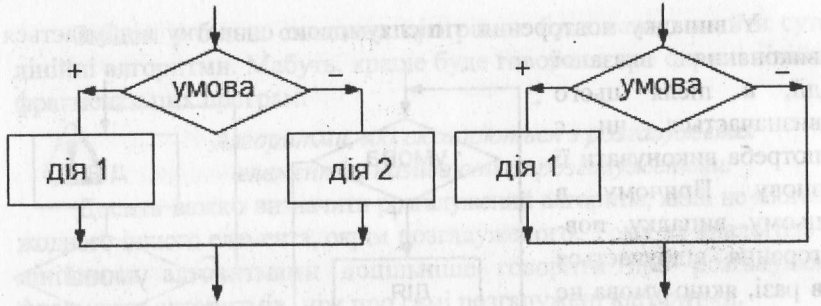
До складених базових алгоритмічних структур відносяться розгалужені і циклічні алгоритми.

Розгалуженим елементом алгоритму називається така операція, за допомогою якої здійснюється вибір однієї з двох можливих дій в залежності від сформульованої умови.

При виконанні операції, яка є розгалуженим елементом, виконується лише одна з дій. В тому випадку, коли умова справджується, продовження виконання алгоритму відбувається за стрілкою «+», в протилежному випадку – за стрілкою «–».

Наведений приклад є повною формою розгалуженого елемента (мал.4), тобто містить вибір однієї з двох передбачених дій.

Скорочена форма розгалуженого елемента у випадку невиконання умови не передбачає ніяких дій (мал.5). При цьому алгоритм переходить до виконання наступного базового елемента (блок «дія 2» буде відсутнім).



Мал.4 Мал.5

Прикладом розгалуженого елемента в алгоритмі Евкліда є вибір зміни значення величини *n* або величини *m* у залежності від співвідношення між ними (*n*>*m* – «так» або «ні»). У результаті вибору одна з дій (*n*:=*n*–*m*, *m*:=*m*–*n*) буде пропущена.

Прикладами розгалужених елементів можуть бути ще такі:

- взяти парасольку за умови, якщо іде дощ (скорочена форма);
- якщо загорілося зелене світло, то переходити вулицю, в протилежному випадку – зачекати (повна форма);
- якщо електронний варіант тексту не містить помилок, то роздрукувати його (скорочена форма);
- якщо знаменник дробу не дорівнює нулю, то обчислити його значення, у протилежному випадку повідомити про помилку (повна форма).

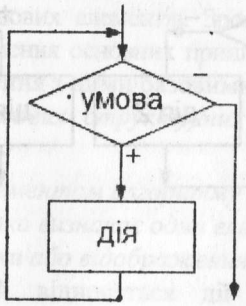
Циклічним елементом алгоритму називається така операція, за допомогою якої здійснюється визначена кількість повторень однієї або декількох дій згідно сформульованої умови.

Більшість алгоритмів містить серії дій, які повторюються декілька разів. Для їх визначення використовують циклічну конструкцію, яка ще носить назву «повторення».

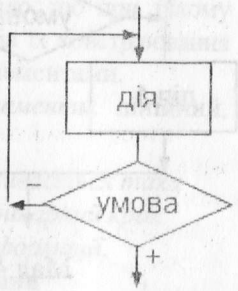
Є два типи повторення: з передумовою (мал.6) та з післяумовою (мал.7).

У першому випадку спочатку перевіряється умова і, якщо вона справджується, то вказана дія черговий раз виконується, якщо ж ні, то виконання дії припиняється.

У випадку повторення з післяумовою спочатку відбувається виконання вказаної дії, а після цього визначається, чи є потреба виконувати її знову. Причому в цьому випадку повторення відбувається в разі, якщо умова не справджується.



Мал. 6



Мал. 7

В алгоритмі Евкліда ми спостерігаємо циклічність під час багаторазового повторення перших трьох дій, поки нові числові значення не стануть однаковими.

До прикладів повторення з передумовою можна віднести такі:

- поки є тісто виготовляти тістечка;
- поки не втомився кидати цеглу у вантажівку;
- поки не завершиться список чисел знаходити їх суму.
- Прикладами повторення з післяумовою можуть бути:
- повторювати виконання фізичної вправи доки вчитель не дасть команду про її завершення;
- продовжувати записи у зошиті до того часу, поки на закінчаться в ньому аркуші;
- продовжувати обчислення швидкості руху при різних значеннях відстані та часу доки результат не набуде наперед заданого значення.

Тепер перейдемо до визначення типів алгоритмів.

Лінійними алгоритмами називаються алгоритми, які складаються з лінійних елементів.

Лінійні алгоритми складаються лише з лінійних елементів, які характерні тим, що після їх виконання виконавець завжди переходить до виконання наступного за порядком елементу в записі алгоритму.

У якості лінійного алгоритму найпростіше, мабуть, навести алгоритм розпорядку робочого дня учня: ранковий підйом, сніданок, заняття у школі, обід, виконання домашніх завдань, відпочинок, вечеря, підготовку до сну, сон.

Згодом у більш складних програмах буде важко знайти суто лінійні алгоритми. Мабуть, краще буде говорити про окремі лінійні фрагменти цих програм.

Алгоритми, які складаються з розгалужених елементів, називаються розгалуженими.

Досить важко визначити розгалужений алгоритм, який не містить жодного іншого елемента, окрім розгалуженого. Тому за аналогією з лінійними алгоритмами доцільніше говорити про розгалужені фрагменти алгоритмів, ніж про самі розгалужені алгоритми.

Серед побутових розгалужених алгоритмів можна навести приклад алгоритму користування парасолькою: якщо почався дощ, то розкрити парасольку, якщо закінчився – закрити.

Алгоритми, що базуються на математичній моделі, переважно використовують аргументи, значення яких необхідно задати, та результати, значення яких треба вивести, які є лінійними елементами. Під час виконання цих частин алгоритму здійснюється діалог з користувачем алгоритму. Переклад з англійської мови слова *interface* означає узгодження, зв'язок. Саме тому ми, говорячи про розгалужені алгоритми, будемо мати на увазі безпосередньо сам алгоритм, а не його інтерфейсні частини.

Наприклад, алгоритм обчислення коренів квадратного рівняння по суті є розгалуженим, оскільки залежить від знака дискримінанта. Однак, перш ніж визначати цю умову, необхідно задати значення коефіцієнтів рівняння, а на останок вивести значення обчислених коренів в разі їх існування, якщо ж ні – то повідомити про це.

Алгоритми, які складаються з циклічних елементів, називаються циклічними.

Необхідність у створенні циклічних алгоритмів виникає, коли треба декілька разів використовувати одні й ті ж формули з різними значеннями змінних, або якісь інші однотипні дії. Циклічний алгоритм за розмірами набагато менший, ніж той, в якому дії були б повторені стільки разів, скільки їх необхідно виконати.

І знову ж таки повторимося, зауваживши, що суто циклічні алгоритми зустрічаються не так часто. Швидше за все на практиці створення алгоритмів можна говорити про циклічні фрагменти алгоритмів. Однак, спробуємо все ж таки навести декілька прикладів циклічних алгоритмів:

- плетіння деякого складного узору за допомогою гачка чи спиць;
- дотримання розпорядку дня протягом робочого тижня;
- заповнення таблиці значень *sinx* при заданих аргументах.

Допоміжними називаються алгоритми, які наперед створені і викликаються на виконання та цілком виконуються в даному алгоритмі тоді, коли виникає в цьому потреба.

Набувши певного досвіду в алгоритмізації і переходячи до складання досить серйозних алгоритмів, ви обов'язково зіткнетеся з ситуацією, коли необхідно описувати дуже схожі фрагменти алгоритму. Звичайно, що ви дійдете думки якимось чином записати цей фрагмент лише один раз і звертатися до нього стільки разів, скільки в цьому виникне необхідність.

Стосовно допоміжних алгоритмів можна розглядати:

- внутрішні, локальні, що створюються в межах даного алгоритму і доступні для використання тільки у цьому алгоритмі;
- зовнішні, глобальні, які можна використовувати у різних незалежних алгоритмах.

Зовнішні допоміжні алгоритми частіше за все об'єднуються у так звані бібліотеки. Для користування цими бібліотеками повинні існувати інструкції з описом всіх допоміжних алгоритмів, які входять до них, та правила користування ними.

Прикладами застосування допоміжних алгоритмів може бути використання табличних значень різних тригонометричних функцій, які наперед розраховані для різних значень аргументів; використання значень відомих фізичних одиниць для розв'язування задач з фізики тощо.

У великих за об'ємом і складних за умовою алгоритмах варто, мабуть, говорити про тип не всього алгоритму цілком, а лише визначаючи тип окремих його фрагментів.

Питання для самоконтролю:

1 Назвіть всі базові структури алгоритмів і дайте їм визначення.
2 Який елемент алгоритму називається лінійним?
3 Який елемент алгоритму називається розгалуженим?
4 Які є форми розгалуження?
5 Який елемент алгоритму називається циклічним?
6 На які типи поділяються циклічні алгоритми?
7 Які алгоритми називаються лінійними?
8 Які алгоритми називаються розгалуженими?
9 Які алгоритми називаються циклічними?

10 Які алгоритми називаються допоміжними?
11 Якими можуть бути допоміжні алгоритми?
12 Як визначається тип великих за об'ємом і складністю алгоритмів?

6. Практична робота «Складання алгоритмів»[1]

За наведеним сценарієм виконайте завдання по створенню алгоритмів.

1) *Побудувати інформаційну модель відповідно до умови задачі.*
2) *Визначити аргументи, результати та проміжні величини алгоритму.*
3) *Розробити словесний опис алгоритму відповідно до умови задачі.*
4) *Побудувати схему алгоритму відповідно до умови задачі.*
5) *Записати алгоритм мовою псевдокодів відповідно до умови задачі.*
6) *Визначити наявність лінійних елементів в побудованому алгоритмі і вказати їх.*
7) *Визначити наявність розгалужених елементів в алгоритмі і вказати їх.*
8) *Визначити наявність циклічних елементів в побудованому алгоритмі і вказати їх.*
9) *Визначити загальний тип алгоритму або окремих його фрагментів.*
10) *Покроково виконати побудований алгоритм.*
11) *Проаналізувати покрокове виконання побудованого алгоритму.*
12) *На прикладі побудованого алгоритму провести аналіз етапів розв'язування задач на комп'ютері.*

7. Побудова алгоритмів

Метод покрокової деталізації. Структурний підхід до побудови алгоритмів. Модульна побудова алгоритму

При створенні складних алгоритмів застосовується ***метод покрокової деталізації***. На кожному етапі складання алгоритму формуються окремі його частини, що є актуальними на даний момент, а решта заміняється, наприклад, текстовим варіантом опису. На наступних етапах така сама методика застосовується до

---

[1] Завдання для даної практичної роботи наведені у моєму «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Складання алгоритмів»

наступної групи фрагментів алгоритму. Такий підхід приводить до поступової деталізації алгоритму, уточнення як виконуваної, так і інформаційної його структури.

Розглянемо вже знайомий нам приклад алгоритму Евкліда. На першому кроці покрокової деталізації сформуємо дію повторення, яка дасть змогу визначити результат алгоритму:

while n<>m **do**
(заміняємо одне з чисел на відповідну різницю цих чисел);

На наступному кроці оформимо в термінах Паскаль-програми вибір числа, яке необхідно замінити відповідною різницею:

```
while n<>m do
 if n>m
    then (замінюємо число n)
    else (замінюємо число m);
```

Тепер залишилось зробити останній крок у деталізації нашого алгоритму – записати дії заміни відповідних значень n та m:

```
while n<>m do
 if n>m
    then n:=n-m
    else m:=m-n;
```

Звичайно, що наведений приклад є одним з найпростіших, на яких продемонстровано принцип покрокової деталізації.

Зусилля по покращенню якості алгоритмів, ефективності праці програмістів знайшли реалізацію в структурному методі побудови алгоритмів.

Структурний підхід до побудови алгоритмів – це спосіб створення алгоритму з широким використанням допоміжних алгоритмів.

У складних та громіздких задачах такий підхід дозволяє розбити алгоритм на окремі частини – модулі, кожний з яких розв'язує свою самостійну підзадачу. Це дає можливість сконцентрувати зусилля на розв'язуванні кожної підзадачі, що реалізується у вигляді окремої підпрограми. Для кожного такого модуля визначаються свої методи реалізації алгоритму та структура даних, якими він оперує.

Останнім кроком у модульній побудові алгоритмів є об'єднання окремих модулів в єдине ціле. Для цього між модулями повинні встановлюватись зв'язки для передачі інформації від одних модулів до інших: результати виконання одних модулів є вхідною інформацією для інших.

Основною метою структурного підходу до побудови алгоритмів є розробка алгоритму з мінімальними взаємозв'язками між його модулями.

Якщо поставлена перед вами задача є серйозним великим завданням, то важливість розбиття алгоритму на окремі модулі неможливо недооцінити. Особливо ефективна така методика при розробці складних алгоритмічних систем колективами програмістів. В цьому випадку кожному розробнику дається своя цілісна частина алгоритму, яка реалізується ним у вигляді окремого модуля. Завершенням колективної роботи є зведення цих модулів в єдине ціле.

Розробка алгоритмів «зверху-донизу» та «знизу-вгору»

Масове застосування комп'ютерів – це в першу чергу розробка нового програмного забезпечення. Тому сьогодні можна вже говорити про сформовану індустрію виготовлення програм. З цією метою створюються різноманітні технології програмування, які ввібрали в себе ідеї як структурного, так і модульного підходу до складання алгоритмів. Познайомимося з найпоширенішими з них.

Програмування «зверху – донизу» або низхідне програмування, базується на ідеї поступової деталізації задачі на ряд окремих підзадач. Спочатку глобальна задача представляється у вигляді набору підзадач і будується програма, в якій ці підзадачі виступають як деякі іменовані підпрограми, до яких можна організувати звернення. Ті підзадачі, які ще програмно не реалізовані, тимчасово заміняються так званими програмними «заглушками», що дозволяє отримати перший варіант програми, готової для першого кроку налагодження і пошуку покращених варіантів. Після цього по відношенню до підзадач застосовується такий самий прийом.

Програмування «знизу – вгору» або висхідне програмування, засноване на протилежному процесі. Спочатку пишуться і налагоджуються частини задачі, які розв'язують окремі під задачі, тобто програми найнижчого рівня, а потім поступово з них збираються крупніші блоки. Цей процес завершиться тоді, коли вся програма буде повністю зібрана і налагоджена.

З часом з'явилася ще одна технологія програмування, що отримала назву технології пакетів прикладних програм. Її ідея полягає в тому, що кожна нова задача розв'язується шляхом комбінації складових частин вже готових налагоджених пакетів програм.

Пакетна технологія знайшла своє продовження і розвиток в об'єктно-орієнтованому програмуванні. Об'єктно-орієнтовані засоби являють собою інструментарії, мови високого рівня, побудовані з модулів, які виконують певні змістовні функції самостійно. Існують інтелектуальні модулі обслуговування екрана, формування меню, підготовки відповідей, контролю помилок при введенні даних користувачем і т. п. Окрім того, в інтелектуальні модулі можуть входити об'єкти, створені самим користувачем або програмістом для подальшої їх обробки і представлення.

Наприклад, поширеність програмних продуктів фірми MicroSoft призвело до певної стандартизації інтерфейсних засобів прикладних програм у вигляді вікон, ієрархічної структури меню користувача тощо.

На сьогоднішній день розроблені і продовжують розроблятися нові об'єктно-орієнтовані інструментальні засоби – середовища, в яких реалізовані мови програмування, що в собі також містять можливості об'єктного програмування.

Методика об'єктно-орієнтованого програмування лежить в основі розробки нових прикладних систем та додатків. Зараз майже кожна традиційна мова програмування розробляється з підмножиною або надмножиною засобів підтримки об'єктного програмування. Наприклад, такі нові системи програмування, як Visual Basic, Visual C++, Visual FoxPro, середовища Delphi, C Builder. Детальніше мова про ідеологію об'єктно-орієнтованого програмування буде йти далі.

Аналіз алгоритмів

Після розробки і складення алгоритму наступає етап його аналізу.

Аналіз алгоритму – це перевірка його на правильність, тобто визначення коректності застосованої математичної моделі, узгодження окремих блоків-підпрограм, логічності виконання всіх дій.

Для аналізу алгоритму застосовується система тестування, про яку мова йшла раніше. Сутність її полягає у підборі таких початкових даних, для яких результат, що має бути отриманий після застосування обраної математичної моделі, є заздалегідь відомим. Наступна звірка отриманих результатів виконання складеного алгоритму з тими ж початковими даними дасть можливість визначити його коректність. Під час перевірки роботи алгоритму послідовно, покроково перевіряється робота окремих його частин – розгалужень, циклів, підпрограм, введення та виведення даних.

Саме при покроковому виконанні алгоритму можна виправити логічні помилки.

Для складних і громіздких алгоритмів може застосовуватись методика використання «заглушок», тобто замість перевірки виконання окремих частин алгоритму тимчасово використовуються значення, які мають бути отримані в результаті їх виконання. Пізніше, після аналізу основних частин алгоритму, можна повернутися до пропущених частин і перевірити їх коректність.

Зрозуміло, що покрокове виконання для громіздких алгоритмів являє собою непросту задачу. Частіше за все «ручний» режим аналізу алгоритмів можна рекомендувати вам виконувати для тих задач, що розміщені в даному посібнику. Для складних алгоритмів сучасні середовища програмування надають користувачеві дуже зручні і корисні можливості щодо покрокового виконання програми за допомогою комп'ютера з відслідковуванням всіх необхідних проміжних значень.

Послідовне уточнення алгоритму

Під час аналізу алгоритму може виникнути необхідність уточнення деяких його частин, заміна деяких фрагментів математичного апарату більш раціональними, оформлення окремих блоків алгоритму у вигляді допоміжних алгоритмів для можливого наступного їх використання в інших його блоках тощо.

Для виконання деяких алгоритмів одним із суттєвих факторів є час їх виконання. При реалізації таких алгоритмів на комп'ютері час виконання алгоритму може бути протестований для різних початкових даних. Подекуди відчутний за часом результат може бути отриманий на малих і великих за об'ємом вхідних даних. Якщо час виконання не влаштовує, то розробник алгоритму намагається застосувати більш ефективні методи розв'язування поставленої задачі. Подекуди вони бувають складнішими в реалізації, але більш ефективними щодо часу їх виконання. В цьому і полягає рівень професійності програміста.

Питання для самоконтролю:

1 *Який підхід до побудови алгоритмів називається структурним?*
2 *Що таке «метод покрокової деталізації»?*
3 *В чому полягає суть розробки алгоритмів «зверху – донизу»?*
4 *В чому полягає суть розробки алгоритмів «знизу – вгору»?*
5 *В чому полягає ідея розробки пакетів прикладних програм?*
6 *Для чого потрібен аналіз алгоритмів?*
7 *Коли виникає потреба в послідовному уточненні алгоритму?*

8. Тематична атестація «Інформаційна модель. Алгоритми»

Для виконання тематичної атестації необхідно:

знати

- етапи розв'язування задач за допомогою комп'ютера;
- різновиди моделей, поняття інформаційної моделі;
- означення алгоритму, класи алгоритмів, означення властивостей алгоритмів;
- поняття виконавця алгоритму;
- означення аргументів, результатів та проміжних величин;
- всі способи описування алгоритмів;
- базові структури алгоритмів, типи алгоритмів;
- суть методу покрокової деталізації алгоритму;
- поняття структурного підходу до побудови алгоритму;
- принцип модульної побудови алгоритму;
- поняття програмування «зверху – донизу» та «знизу – вгору»;
- необхідність послідовного уточнення алгоритму;
- способи здійснення аналізу алгоритму;

вміти

- визначати послідовність етапів розв'язування задач за допомогою комп'ютера;
- пояснювати принципи побудови інформаційної моделі;
- наводити приклади алгоритмів;
- обґрунтовувати відмінність між аргументами, результатами та проміжними величинами;
- наводити схему роботи алгоритму;
- визначати відмінність різних способів опису алгоритмів;
- наводити власні приклади опису алгоритмів;
- пояснювати особливості кожної з базових структур алгоритмів;
- визначати тип алгоритму;
- складати алгоритми відповідно до умови задачі;
- наводити власні приклади різних типів алгоритмів;
- на власному прикладі пояснювати суть методу покрокової деталізації алгоритму;

- пояснювати основну мету структурного підходу до побудови алгоритму;
- наводити приклади модульного підходу до побудови алгоритму;
- пояснювати необхідність послідовного уточнення алгоритму;
- обґрунтовувати необхідність аналізу алгоритму.

Програма. Мова програмування

9. Поняття програми

Поняття про мови програмування

При підготовці алгоритмів для їх виконання на комп'ютері на перший план виходить необхідність точного запису команд, зрозумілих виконавцю, що не залишає місця для довільного їх трактування.

Мовою програмування називається фіксована система позначень для опису алгоритмів і структур даних.

Мова більшості сучасних комп'ютерів досить «скупа» і складається з команд-наказів типу «виділити пам'ять певного розміру», «прочитати інформацію з певного місця пам'яті», «запам'ятати інформацію у певному місці пам'яті», «додати два числа», «перейти до виконання наступної команди, прочитавши її з певного місця пам'яті», «порівняти два числа» тощо. Як правило, команд всього декілька сотень. Всі вони настільки прості, що можуть бути ефективно реалізовані апаратурою комп'ютера. Набір цих команд, що носить назву мови машинних кодів, з точки зору їх функціональності є повним. Використовуючи команди з цього набору, тобто користуючись заданою системою команд, можна описати будь-який алгоритм. Однак такий запис для складних задач настільки громіздкий, що у людини буде дуже мало шансів зробити його безпомилковим.

Існує два виходи з цієї ситуації. Перший пов'язаний із створенням більш складних комп'ютерних систем, тобто з апаратним забезпеченням, що зможе реалізувати значно ширший і складніший набір команд. Другий шлях – це розробка програмного забезпечення, яке б дозволяло використовувати складні з точки зору комп'ютера, але більш зрозумілі з точки зору людини команди, як, наприклад, розгалуження та цикли. Але при цьому виникає необхідність створення спеціальних програм-перекладачів, функцією

яких є перетворення програм із вхідної мови програмування в еквівалентні програми, що може виконати комп'ютер. Такі програми називаються трансляторами, мова про них буде йти далі. Саме другий шлях отримав активний розвиток.

На сьогоднішній день розроблена чимала кількість різних мов програмування. Окрім того ще відомі різні версії одних і тих самих мов програмування. Чим пояснюється така їх кількість? Чому не можна обійтися однією або принаймні декількома? Справа в тому, що на сьогоднішній день сфери використання комп'ютерів такі різноманітні і такі специфічні, що врахувати усі в одній мові просто неможливо. Тому кожна мова програмування орієнтована на певний клас задач. Для програмування економічних задач використовуються мови програмування dBase, Paradox, Clipper; для розрахункових задач досить скористатися можливостями мов Basic, Pascal; для доступу до більш «глибоких» можливостей комп'ютера бажано скористатися можливостями мов Асемблер та Сі.

Класифікація мов програмування

Існують різні класифікації мов програмування. Згідно найпоширенішої з них всі мови програмування поділяються на мови низького та високого рівня.

Мови машинних команд (кодів) даної моделі комп'ютера, які сприймаються ним безпосередньо, називаються мовами низького рівня або машинними мовами.

До групи мов програмування низького рівня відносяться:

- ***мова мікрокоманд***, яка задає найпростіші передачі даних між оперативною пам'яттю та процесором, між самими регістрами (комірками) процесора тощо;
- ***машинна мова***, кожна команда якої описується послідовністю мікрокоманд;
- ***асемблер*** – мова символічного кодування.

Перші дві мови погано налаштовані на використання людиною, оскільки їх команди задаються послідовністю нулів та одиниць. Оператори асемблера – це ті ж машинні команди, але вони мають символічні (мнемонічні) назви, а в якості операндів використовуються не конкретні адреси комірок оперативної пам'яті, а їх імена.

Всі мови програмування низького рівня орієнтовані на певний тип комп'ютерів і в цьому смислі спеціалізовані саме для них.

Мовами програмування високого рівня називаються мови, на яких програми складаються з операторів.

Для всіх мов високого рівня загальним є те, що вони орієнтовані не на систему команд того чи іншого комп'ютера, а на загальну систему операторів, характерних для запису певного класу алгоритмів. Типовими прикладами операторів таких мов можуть бути оператори присвоювання, умовні оператори, оператори циклів тощо. Перш ніж комп'ютер виконуватиме програму, написану мовою високого рівня, необхідно перекласти її на машинну мову, яка відповідає даному типу комп'ютера. Для цього призначені спеціальні програми – транслятори, про які детальніше розмова йтиметься далі.

Мови програмування Basic, Pascal та Сі називаються мовами програмування високого рівня, тому що їх конструкції максимально наближені до звичайних розмовних мов і дуже зручні та зрозумілі людині.

Ми розглянули одну із традиційних класифікацій мов програмування, в основі якої лежить виразна потужність мови. Однак існують класифікації і за іншими ознаками. Одна з них – це класифікація, відповідно до якої мови програмування поділяються на обчислювальні та логічні. Практично всі розглянуті вище мови програмування є обчислювальними. Якщо ж говорити про логічні мови, то слід згадати про Ліспі (List Processing Language), Пролог (Programming of Logic).

Мова Ліспі створена в 1965 р. американським професором Джоном Маккарті для дослідження проблем штучного інтелекту і стала основою ряду програмних реалізацій інтелектуальних систем. На відміну від Ліспа Пролог – це європейська мова, яка базується на логічному численні. Її розробив А.Колмерауер у 1972 р. в Марсельському університеті.

Існує також і «прикладна» класифікація мов програмування, яка вже була розглянута вище. Згідно з цією класифікацією всі мови програмування поділяють на групи за галузями застосування.

Процедурна мова програмування

Більшість мов програмування високого рівня відноситься до процедурних мов. В основному це мови програмування для розв'язування обчислювальних задач, а саме Пейсик, Паскаль, Сі тощо.

Мови програмування, в яких реалізована можливість покрокової деталізації алгоритму, низхідного та висхідного програмування, називаються процедурними мовами.

В зазначених вище мовах основним методом є розбиття задачі на окремі кроки та їх послідовний опис. Покрокова деталізація алгоритму відбувається до того моменту, поки ми не отримаємо елементарні дії розв'язування окремих фрагментів задач, які можна буде записати у вигляді команд, допустимих в даній мові.

Ще одна особливість процедурних мов програмування полягає в тому, що команди в програмах, записаних процедурними мовами, виконуються послідовно. Це означає, що результати виконання всіх попередніх операцій вже отримані і зворотної сили вони не мають. Оскільки вміст пам'яті, де зберігається вся проміжна інформація виконуваних операцій, постійно поновлюється і попередня інформація не зберігається, то відмінити раніше виконані команди неможливо. Мови програмування такого виду ще називаються алгоритмічними.

В основі процедурних мов програмування закладені принципи структурного програмування. Розглянемо відомий вже нам приклад фрагмента алгоритму пошуку найбільшого спільного дільника:

```
while x<>y do
 if x>y
 then
    x:=x-y
 else
    y:=y-x;
```

Наведений запис зовні являє собою одну дію – повторення типу «поки» (***while***). Але цей зовнішній оператор містить в собі підпорядковані оператори, що міняють значення x або y. Така ієрархія операторів відображає структуру, що лежить в основі алгоритму. Її наочно видно завдяки структурі мови програмування, що допускає вкладеність компонентів програми один в один, використання їх послідовно один за одним. Алгоритми, які ми отримуємо в

результаті, мають чітку і ясну структуру. В тексті програми така вкладеність чи підпорядкованість відображається зсувами рядків.

Ключовою ідеєю структурного побудови алгоритму є відображення структури алгоритму в структурі тексту програми.

Справді, неможливо зрозуміти смисл програми, якщо зникне її структура, як це буває, коли компілятор видає машинний код. Ми повинні розуміти, що програма некорисна, якщо людина не може в ній розібратися і пересвідчитися в її правильності.

Логічне програмування

Ідея використання математичної логіки в якості мови програмування була запропонована Р.Ковальським в Англії на початку 70-х років XX століття. Логічне програмування виникло в основному завдяки успіхам в автоматичному доведенні теорем.

Основною метою математичної логіки є забезпечення певної системи формальних позначень для відображення міркувань. Одним з розділів математичної логіки є логіка висловлювань.

Логіка висловлювань – це розділ математичної логіки, який вивчає висловлювання, що розглядаються з боку їх логічних значень – істинності та хибності – і логічних операцій над ними.

Основним елементом логіки висловлювань є висловлювання і операції над ними. В логіці висловлювань є тільки стверджувальні речення, які можуть бути істинні або хибні, а не і те і інше одночасно. Кожне таке стверджувальне речення називається висловлюванням. Наприклад, «Петренко є учень», «Іваненко вивчає інформатику». Із простих висловлювань можна будувати складені. Наприклад, «Петренко є учень і ходить до школи», «Якщо Іваненко ходить до школи, то він вивчає інформатику».

В логіці висловлювань використовується п'ять основних логічних операцій:

- «Ні» – заперечення;
- «І» – кон'юнкція;
- «Або» – диз'юнкція;
- «Якщо ..., то ...» – висновок (імплікація);
- «Тоді і тільки тоді» – еквівалентність.

Логічне програмування – це запис задачі мовою математичної логіки у вигляді набору висловлювань, відношень між висловлюваннями та правил виведення одних висловлювань з інших.

В результаті задача, записана в термінах логічного програмування, перевіряється на справедливість сформульованих висловлювань.

Як зазначалося вище, основні ідеї логічного програмування вперше були успішно реалізовані А. Колмерауером у Франції у формі мови програмування Пролог. Терміни «логічне програмування» та «програмування мовою Пролог» часто використовують як ідентичні.

Пролог був прийнятий в якості базової мови в японській програмі створення комп'ютера п'ятого покоління, що орієнтована на дослідження методів логічного програмування та штучного інтелекту, а також на розробку нового покоління комп'ютерів, спеціально призначених для реалізації даних методів. Назва мови Пролог утворена із слів ПРОграмування ЛОГічного або ПРОграмування в термінах ЛОГіки (PROLOG – PROgramming in LOGic).

Логічне програмування мовою Пролог докорінно відрізняється від програмування алгоритмічними мовами. Програма мовою Пролог описує не процедуру розв'язування задачі, як це має місце в традиційних алгоритмічних мовах типу Бейсик, Паскаль, Сі і т.п., а логічну модель предметної області, що включає в себе деякі факти відносно властивостей предметної області і відношень між об'єктами в цій області, а також правила виведення нових властивостей і відношень із вже заданих.

Програма мовою Пролог складається із множини тверджень, кожне з яких є або фактом, або правилом і одного цільового твердження.

Правило вказує, яким чином розв'язок пов'язаний із заданими фактами або як можна із заданих фактів вивести цей розв'язок.

Програмування мовою Пролог включає в себе такі етапи:

- оголошення фактів про об'єкти і відношення між ними;
- визначення правил взаємодії об'єктів і відношень між ними;
- формулювання питань про об'єкти і відношення між ними.

Наведемо приклад. Нехай є такі твердження:

«Іваненко – це учень»;

«Сидоренко є однолітком Іваненка»;

«Петренко є однолітком Сидоренка»;

«Об'єкт є учнем за умови, що він є однолітком учня».

Перші три елементи даних є фактами, четвертий – правилом. Якщо ввести таке цільове твердження: «Учнями є:», то отримаємо відповідь, що Іваненко, Сидоренко, Петренко – учні, причому Іваненко є учнем за визначенням, а Сидоренко та Петренко – внаслідок виведення.

Отже, програма мовою Пролог дуже схожа на гіпотезу про деяку предметну галузь, а питання – на теорему, яку необхідно довести.

Підведемо підсумок. На відміну від процедурного програмування, програма мовою Пролог не виконується послідовно по кроках, оскільки не задає ніяких дій. Вміст Пролог-програми – це сукупність фактів та зв'язків між ними. Тому програмування на Пролозі полягає у передачі комп'ютеру сукупності інформації та наступному зверненні до нього з питаннями щодо можливих висновків із цієї інформації. І ще одна можливість: логічну програму можна використовувати для пошуку будь-якої інформації, яка може бути логічно виведена із даної програми. Пошук об'єкта, що знаходиться у заданому відношенні з іншими об'єктами, є дуже важливою властивістю логічних програм.

Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (скорочено ООП) – це в наш час природний сучасний підхід до побудови складних програм і систем.

В основі об'єктно-орієнтованого програмування лежить ідея об'єднання в одній структурі даних і дій, які виконуються над ними. В термінології ООП ці дії називаються методами. При такому підході організація даних і програмна реалізація дій над ними виявляються значно сильніше пов'язаними, ніж при традиційному структурному програмуванні.

Об'єктно-орієнтоване програмування базується на трьох основних поняттях: інкапсуляції, наслідуванні, поліморфізмі.

Інкапсуляція – це комбінування даних з процедурами і функціями, які маніпулюють цими даними.

Інкапсуляцію можна уявити собі, як механізм, що об'єднує дані і процедури, які їх обробляють, в єдину структуру. В ООП кажуть, що таким чином створюється «чорний ящик», в якому сховано механізм представлення та обробки даних. Користувач може звертатися до даного об'єкту за заздалегідь визначеним правилом і, не впливаючи на об'єкт, лише користуватися ним.

Схожий механізм використовується і у стандартних бібліотеках мов програмування, коли ми звертаємося до функції, користуємося нею, не задумуючись над тим, яким саме чином вона працює.

Наслідування – це можливість використання вже визначених об'єктів для побудови ієрархії об'єктів, похідних від них.

Наслідування – це механізм, в результаті застосування якого один об'єкт може отримувати у спадковість всі властивості та методи іншого, додаючи до них свої характерні властивості. Кожний з «нащадків» отримує у спадковість опис даних «прабатька» і доступ до методів їх обробки. Таким чином з'являється ієрархія.

В якості прикладу можна навести такий ланцюжок: *рослина ⇨ овоч ⇨ морква*. В даному випадку овочу притаманні всі властивості рослини, але додаються і свої особливості, а морква є овочем, але її властивості дещо відмінні від картопляних і т. д.

Поліморфізм – це можливість визначення єдиної за іменем дії (процедури або функції), що може бути застосована одночасно до всіх об'єктів ієрархії наслідування.

Поліморфізм представляє собою механізм, що дозволяє одне і те ж ім'я використовувати для розв'язку схожих задач, які все ж таки в окремих деталях відмінні.

В якості прикладу можна звернутися до мови програмування Сі, де для знаходження модуля числа використовуються три різні функції: abs() – для цілих чисел, labs() – для довгих цілих, fabs() – для дробових. У відповідній моделі ООП використовуємо одну функцію з назвою abs(), що, в залежності від типу вхідних даних, сама визначає, який з трьох алгоритмів застосувати. Ще один приклад – це додавання. I 'a'+'x' , і 1+2 – операція додавання, але у двох випадках реально виконуються різні дії, хоча від користувача цей механізм прихований.

Прикладом застосування об'єктно-орієнтованого програмування є операційна система Windows. Коли ви відкриєте будь-яку програму Windows, то побачите вікно з великою кількістю кнопок, розділів меню, вікон редагування, списків тощо. Все це об'єкти. Причому самі по собі вони нічого не роблять, а чекають якихось подій, наприклад, натиснення користувачем клавіш або кнопок миші, переміщення курсора і т. і. Коли відбувається подібна подія, об'єкт отримує повідомлення про це і певним чином на нього

реагує: виконує деякі обчислення, розгортає список, заносить символ у вікно редагування.

Об'єктно-орієнтоване програмування – це нова ідеологія програмування, що базується на використанні сукупності об'єкта та подій, на які він може реагувати.

Об'єктно-орієнтований підхід може помітно спростити написання складних програм, надати їм гнучкості. Однією з його головних переваг можна назвати можливість розширення області їх застосування, не переробляючи самі програми, а лише додаючи в них нові рівні ієрархії.

Візуальне програмування

Логічною реалізацією об'єктно-орієнтованого програмування є візуальне програмування.

Включати об'єкти в свою програму можна двома способами:

- вручну, використовуючи в програмі відповідні оператори, що насправді відбувається досить рідко;
- шляхом візуального програмування, використовуючи заготовки середовища – компоненти.

Ще недавно розробка графічного інтерфейса користувача була пов'язана з великими зусиллями. Програмування вручну різних звичних користувачу вікон, кнопок, меню, обробка натиснення клавіш на клавіатурі та миші, включення в програми зображень і звуку вимагало все більше і більше часу програміста. В багатьох випадках весь цей сервіс займав до 80-90% об'єму програмних кодів. При зміні стилю графічного інтерфейса вся ця праця ставала марною, все доводилось починати спочатку.

Вихід з цієї ситуації було знайдено завдяки двом підходам. По-перше, було стандартизовано багато функцій інтерфейса, завдяки чому з'явилась можливість використовувати готові бібліотеки, які є, наприклад, у Windows. По-друге, з'явилося візуальне програмування, що дозволило звести проектування користувацького інтерфейса до простих і наочних процедур, які дають можливість за хвилини або години зробити те, на що раніше йшли місяці роботи.

В інтегрованому середовищі розробки Delphi це виглядає дуже просто. Середовище надає користувачу форми, на яких він може розміщувати компоненти, наприклад, кнопки. Форма частіше за все візуалізується на екрані монітора у вигляді вікна, яке потім і

буде бачити користувач програми. На форму за допомогою миші переносяться і розміщуються піктограми компонентів, що є в наявності в бібліотеці Delphi.

За допомогою простих маніпуляцій можна змінити розміри, розташування та властивості цих компонентів. Весь цей процес проектування візуалізується на екрані. Головне полягає в тому, що Delphi під час проектування форми і розміщення на ній компонентів автоматично формує коди програми, що описують даний компонент. До використаних компонентів можна приєднати обробників подій, тобто власні програмні коди, що передбачають реакцію програми на роботу з вказаним компонентом.

Візуальне програмування – це практичне застосування ООП при використанні готових бібліотек компонентів, передбачених середовищем програмування.

Поняття про системи програмування

Всі розглянуті ідеологічні аспекти програмування реалізовані в конкретних мовах програмування. Але кожна мова розробляється для її подальшої реалізації на комп'ютері. Саме для такої реалізації конкретних мов з їх алфавітом, правилами запису окремих дій і розробляються системи програмування. Системи програмування входять до складу математичного забезпечення комп'ютера.

Система програмування – це реалізація конкретної мови програмування для певних комп'ютерних систем.

Кожна мова програмування підтримана своєю системою програмування з урахуванням можливостей конкретної операційної системи. В комп'ютерах перших поколінь системи програмування мали дуже обмежені можливості, які надавали програмісту лише змогу перевести програму з алгоритмічної мови в машинний код і визначити синтаксичні помилки. Вони не містили навіть текстового редактора, тому доводилось текст програми набивати на перфокарти за допомогою окремих пристроїв – перфораторів.

З часом системи програмування потужнішали, і зараз ми користуємося можливостями редактора текстів при наборі самої програми, збереження як тексту програми, так і її машинного коду, можливостями покрокового виконання програми для визначення логічних помилок та запуску програми на виконання.

Сучасні системи програмування доповнені розвиненим користувацьким інтерфейсом, який реалізований у вигляді інтегрованих інтерактивних середовищ (ІІС) або інтегрованих інструментальних оболонок (ІІО). Такі середовища переважно забезпечують користувачу багатовіконний та багатофайловий режим роботи, використання миші, дозволяють застосовувати об'єктно-орієнтоване програмування, використовувати фрагменти програм, написаних на асемблері.

З інтегрованим інтерактивним середовищем Turbo Pascal 7.0 ми детальніше будемо знайомитись пізніше.

Поняття про інтерпретацію та компіляцію

На прикладі двох мов програмування Basic та Pascal цікаво розглянути питання про перетворення програм, написаних цими мовами, у виконуваний вигляд, тобто набір машинних кодів. Для цього існують спеціальні системні програми – транслятори (у перекладі з англійської мови translate – «перекладати»).

Будь-який транслятор виконує дві основні задачі.

Перша – аналіз програми, що транслюється, в результаті чого визначається її коректність. При виявленні помилок транслятор вказує на ті місця тексту програми, де порушені правила її написання.

Друга – генерація вихідної програми мовою команд комп'ютера.

Транслятори є двох типів – інтерпретатори та компілятори.

Інтерпретатори перекладають по одній команді або оператору вхідної програми на машинну мову і відразу ж виконують їх.

Інтерпретатор мови програмування Basic здійснює послідовний синтаксичний контроль операторів початкового тексту програми і виконання її команд (операторів).

Компілятори перекладають всю програму, написану мовою програмування високого рівня, на машинну мову, після чого програма записується в оперативну пам'ять і виконується

Компілятор мови програмування Pascal здійснює переклад всього початкового тексту програми в машинний код.

Отже, Basic-програма на етапі виконання буде виконуватися по одній команді. Тому навіть при наявності синтаксичних помилок в тексті та частина програми, яка передує їм, все одно буде

виконана. Pascal-програма буде виконуватися лише після виправлення всіх синтаксичних помилок в початковому тексті.

Результат компіляції програми, тобто її машинний код, можна зберегти на зовнішньому носієві у вигляді окремого файла. Таку відкомпільовану програму можна експлуатувати без використання середовища програмування. На жаль, не всі інтерпретатори надають таку можливість. Для повторного виконання програми, написаної мовою програмування високого рівня, необхідно знову запускати інтерпретатор.

Питання для самоконтролю:

1 *Що називається мовою програмування?*
2 *Чим викликана значна кількість різних мов програмування?*
3 *На які класи традиційно поділяються мови програмування?*
4 *За якими ще ознаками класифікуються мови програмування?*
5 *Які мови програмування називаються процедурними?*
6 *Що таке структурне програмування?*
7 *На якому розділі математики базується логічне програмування?*
8 *Що таке «логіка висловлювань»?*
9 *Яке програмування називається логічним?*
10 *Яка мова програмування втілила в себе ідеї логічного програмування?*
11 *Що називається об'єктно-орієнтованим програмуванням?*
12 *На яких трьох основних поняттях базується об'єктно-орієнтоване програмування і в чому їх суть?*
13 *Що таке «візуальне програмування»?*
14 *Що називається системою програмування?*
15 *Яку функцію виконують транслятори?*
16 *Поясніть різницю між інтерпретацією та компіляцією.*

10. Середовище програмування

Інтегровані середовища програмування. Поняття редактора, транслятора, налагоджувача

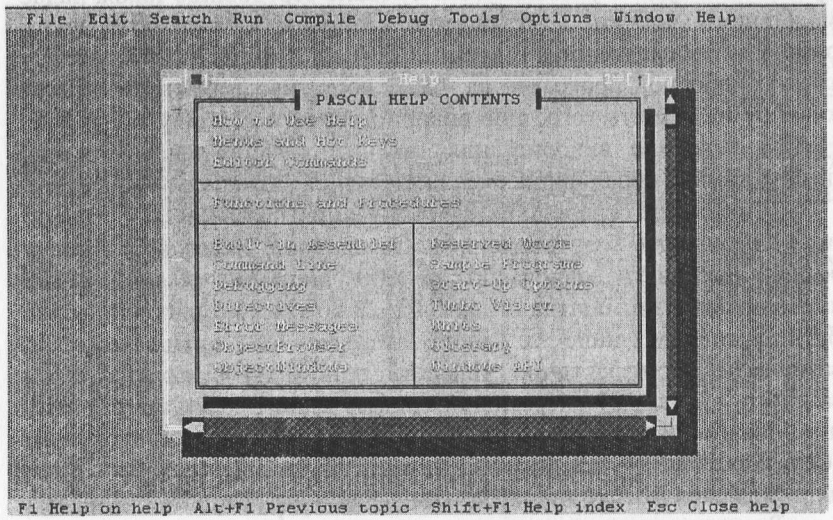
Здається, ось-ось у вас виникне дуже слушне запитання: як виконати написану програму на Паскалі?

Давайте для прикладу візьмемо досить зрозумілу програму і на розглянемо повний алгоритм її реалізації, починаючи від набирання тексту в середовищі програмування до отримання результату її виконання.

Нехай це буде програма, яка за введеним ім'ям користувача, вітає його з початком вивчення розділу основ алгоритмізації та програмування. Домовимося про те, що на даному етапі ми не будемо вдаватися у деталі пояснення структури цієї програми.


```
program my_first_program;
 var name: string;
 begin
  write('Задайте Ваше ім'я і натисніть клавішу ENTER: ');
  readln(name);
  writeln('Шановний ',name,'!');
  writeln('Вітаю Вас з початком вивчення алгоритмізації!');
  writeln('Для завершення програми натисніть клавішу ENTER');
  readln
 end.
```

Дійсно, текст програми, написаний на аркуші паперу, сам по собі не може бути виконаний. Для зручності роботи користувача з Паскаль-програмами створено інтерактивне інтегроване середовище, яке об'єднало в собі можливості текстового редактора для набирання текстів програм, компілятора для визначення помилок у програмах та запуску програм на виконання в разі відсутності помилок, налагоджувача для покрокового виконання програм і визначення складних помилок. Інтерактивним середовище називається тому, що воно працює в режимі постійного спілкування з користувачем, а інтегрованим тому, що об'єднує в собі одночасно всі згадані вище можливості.



Мал.8

Робота в інтегрованому середовищі починається із запуску на виконання файла **turbo.exe**. Окрім цього файла на вашому диску бажано мати в наявності ще такі файли:

- **turbo.tpl** – бібліотека стандартних процедур та функцій Turbo Pascal 7.0;
- **turbo.hlp** – допомога користувачу середовища програмування Turbo Pascal 7.0.

Після запуску середовища Turbo Pascal 7.0 (мал.8) у верхній частині екрана монітора бачимо головне меню, а в нижній – рядок повідомлень. Можливості середовища Turbo Pascal 7.0 дуже потужні і розглядати їх всі в межах даного посібника немає ніякого сенсу. Тому зупинимося лише на основних та необхідних моментах.

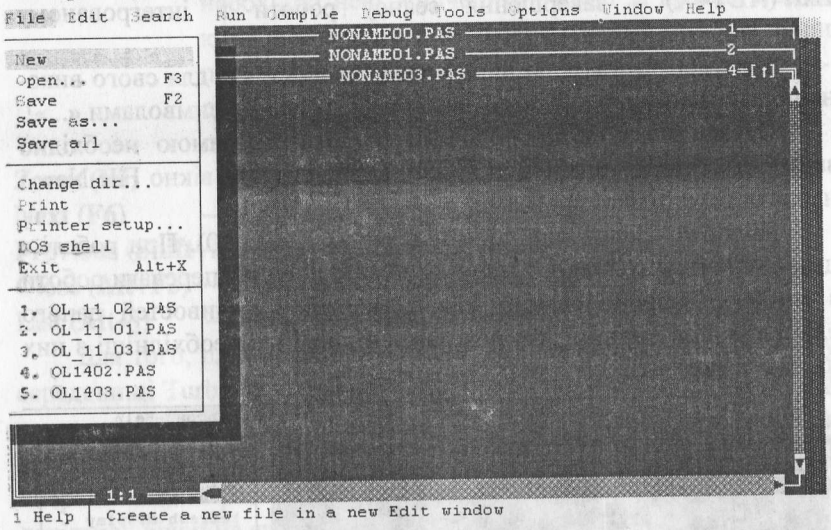
Середовище Turbo Pascal 7.0 дозволяє користувачу працювати як за допомогою клавіатури, так і за допомогою миші. Для роботи з клавіатурою в назві кожного елемента головного меню іншим кольором виділені окремі літери, за допомогою яких можна активізувати те чи інше меню. Для цього достатньо натиснути комбінацію клавіш **ALT+«літера»**. Наприклад, для активізації меню **File** необхідно скористатися комбінацією клавіш **ALT+«F»**. Якщо ж на вашому комп'ютері інстальована миша, то достатньо підвести її курсор до необхідного елемента головного меню і натиснути на ній будь-яку клавішу. Інтегроване середовище також надає можливість роботи з «гарячими клавішами», які активізують роботу найбільш використовуваних команд. Наприклад, для запису тексту набраної програми на диск достатньо натиснути гарячу клавішу **F2** замість того, щоб використовувати команду меню **File**. Надалі можливе використання гарячих клавіш для виконання деяких команд середовища буде вказуватися в дужках.

Інтегроване середовище програмування Turbo Pascal 7.0 ще по праву називають багатовіконним. Середовище надає користувачу змогу працювати одночасно із 100 вікнами, в яких може знаходитися різноманітна інформація. В кожний момент часу може бути активним лише те вікно, в якому ви працюєте. Слід зауважити, що практично кількість одночасно відкритих вікон залежить від об'єму оперативної пам'яті вашого комп'ютера, тому що інформація у вікнах під час сеансу роботи з середовищем зберігається саме в оперативній пам'яті. Лише за вашим бажанням ця інформація може бути збережена у файлі на дискові.

Розглядаючи головне меню, ми дещо змінимо вказаний порядок їх слідування, ставлячи за мету під час ознайомлення на перше

місце логічність пояснення.

File – меню для роботи з файлами, які містять тексти програм (мал.9). Після активізації цього елемента головного меню перед вами розгорнеться меню, яке міститиме такі команди.



Мал.9

New – відкрити нове вікно для створення інформації (автоматично задається ім'я файла NONAMExx.PAS, де xx – порядковий номер відкритого вікна);

Open... (F3) – прочитати з диска для подальшої роботи текст збереженої раніше програми з поточного підкаталога, тобто з того, з якого було запущено середовище;

Save (F2) – зберегти вміст активного вікна в поточному підкаталозі;

Save as... – зберегти вміст активного вікна з новим вказаним ім'ям;

Save all – зберегти вміст всіх відритих вікон;

Change dir... – зміна каталога для читання або запису файлів;

Print – виведення на пристрій друку вмісту активного вікна;

Get info... – інформація про розподіл пам'яті комп'ютера інтегрованим середовищем;

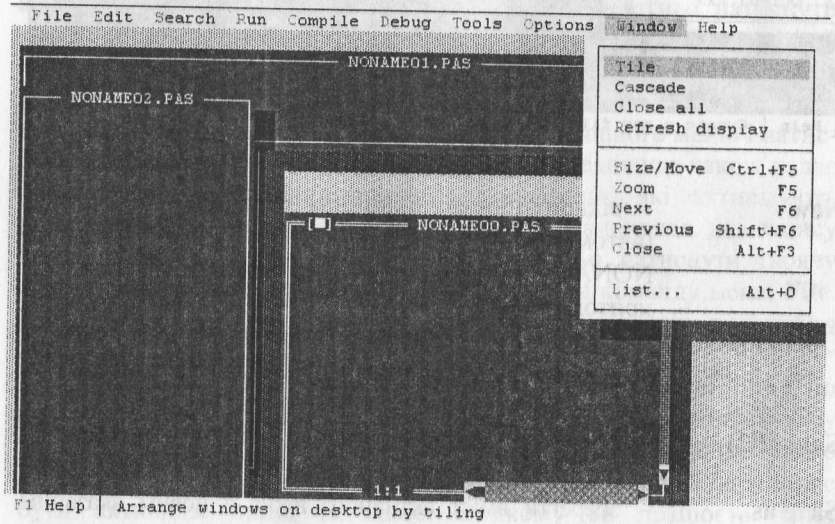
Dos shell – тимчасовий вихід в операційну систему для виконання дій на рівні ОС (повернення назад в інтегроване середовище здійснюється командою EXIT);

Exit (ALT+X) – завершення сеансу роботи в інтегрованому середовищі.

Ви звернули увагу на те, що назви команд, які для свого виконання вимагатимуть додаткової інформації, доповнені символами «...»!

Отож, для початку роботи з нашою програмою необхідно запустити середовище Turbo Pascal, відкрити нове вікно File-New і набрати текст нашої програми.

Window – меню для роботи з вікнами (мал.10). При роботі з цим елементом головного меню особливо відчутні переваги роботи з мишею. Запропонований нижче перелік можливостей даного елемента не є повним. Ми розглянемо лише найнеобхідніші з них на даний момент.



Мал.10

Зверніть увагу на те, що при роботі з багатьма вікнами активне вікно виділяється жирною подвійною рамкою.

Tile – рівномірний розподіл всіх відкритих вікон на екрані монітора;

Cascade – розподіл всіх відкритих вікон у вигляді каскаду, тобто одне за одним;

Close All – закриття всіх активних вікон;

Size/Move (Ctrl+F5) – зміна розмірів та положення активного вікна. Для зміни розмірів вікна за допомогою миші необхідно нею «підчепити» вікно за правий нижній кут і рухати його в необхідному напрямку. Для зміни положення зменшеного вікна на екрані монітора необхідно «підчепити» його мишею за верхню подвійну рамку і рухати в потрібному напрямку;

Zoom (F5) – розгорнути вміст активного вікна на повний екран;

Next (F6) – активізація наступного вікна;

Previous (Shift+F6) – активізація попереднього вікна;

Close (Alt+F3) – закриття активного вікна;

List (Alt+j) – список всіх відкритих вікон.

Для того, щоб переконатися у можливості роботи з вікнами у середовищі Turbo Pascal, змініть розмір вікна, в якому знаходиться текст нашої програми, спробуйте перемістити це вікно по екрану монітора, відкрийте ще декілька нових вікон і виконайте з ними операції зміни розмірів та всі можливі варіанти розташування декількох вікон на екрані монітора.

Run (Ctrl+F9) – запуск програми з активного вікна на виконання з одночасним пошуком синтаксичних помилок.

Для того, щоб виконати набрану програму, перш за все необхідно зробити вікно, в якій знаходиться її текст, активним. Для цього необхідно клацнути «мишею» на верхній рамці цього вікна або перейти до нього за допомогою однієї з операцій Next, Previous або List. Після того, як ви переконаєтесь, що вікно з нашою програмою є активним, натисніть комбінацію клавіш Ctrl+F9. Результатом цього буде запуск програми на виконання. Якщо ви не зробили жодних помилок, то програма почне виконання і вам треба буде виконувати всі вимоги програми: задати ім'я, натиснути клавішу ENTER тощо. У разі наявності помилок у тексті програми, компілятор підкаже їх вам, встановивши курсор в позиції, що слідує за помилкою. Ваші наступні дії в такому разі зрозумілі: подивіться на зразок тексту програми, визначте помилку, виправте її, користуючись типовими правилами редагування текста, і запустіть програму на повторне виконання комбінацією клавіш

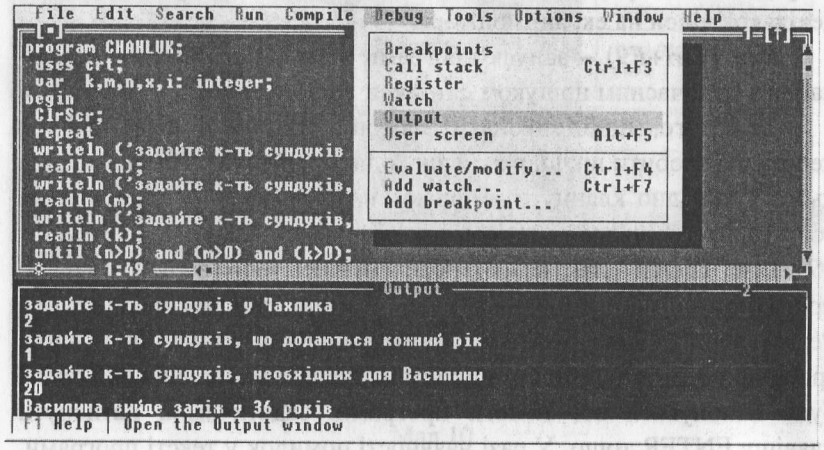
Ctrl+F9. Лише за повної відсутності помилок в тексті програми ви отримаєте результат її виконання.

Compile – компіляція програми з активного вікна.

Compile (Alt+F9) – створення exe-файла програми, яка знаходиться в активному вікні;

Destination Memory (Disk) – компіляція програми в оперативній пам'яті комп'ютера без збереження результату компіляції на диску (із збереженням отриманого в результаті компіляції exe-файла програми на диску в поточному підкаталозі). Щоб перемкнути один режим на інший необхідного на цьому елементі меню, натиснути клавішу миші або клавішу **ENTER**.

Компіляція програми буде вам необхідна у разі, якщо ви захочете пересвідчитися у відсутності в ній помилок без її виконання. Окрім цього дана опція середовища надає можливість збереження виконуваного коду нашої програми на диску. Надалі можна користуватися програмою, не запускаючи середовища програмування. Це може бути корисним у випадку, якщо ваша програма надалі буде неодноразово використовуватись.



Мал.11

Edit – редагування інформації в активному вікні (мал.11). Для більш гнучкого редагування текстів інтегроване середовище використовує спеціальну область пам'яті – «кишеню», яка тимчасово зберігає необхідну інформацію. Виділення фрагмента

тексту, з яким буде вестися подальша робота, виконується за допомогою клавіші **SHIFT** та курсорних клавіш «вгору», «вниз», «вправо», «вліво». Те ж саме можна виконати за допомогою миші, підвівши її на початок фрагмента тексту, натиснувши на кнопку миші і рухаючи її в необхідному напрямку.

Зняти відмітку тексту можна за допомогою послідовного натиснення клавіш **Ctrl+K+H** або натисненням клавіші миші в будь-якому місці відміченого тексту.

Cut (Shift+Del) – вилучення відміченого фрагмента тексту з активного вікна і переміщення його в «кишеню»;

Copy (Ctrl+Ins) – копіювання відміченого фрагмента тексту з активного вікна в «кишеню»;

Paste (Shift+Ins) – копіювання інформації з «кишені» в активне вікно, починаючи з поточної позиції курсора;

Show clipboard – перегляд в окремому вікні вмісту «кишені» і можливість роботи з ним як зі звичайним текстом;

Clear (Ctrl+Del) – знищення відміченого фрагмента тексту.

Для виконання редагування тексту програми, яка була запропонована для прикладу на початку цього заняття, можна в повній мірі використовувати всі запропоновані можливості. Особливо це стане в нагоді, якщо ви працюєте у різних вікнах середовища з декількома програмами. В такому разі можна копіювати фрагменти різних програм, що збігаються, як це ми робили в інших текстових редакторах.

Debug – меню для роботи з налагодженням програми в активному вікні.

Output – відкрити спеціальне службове вікно, в якому можна побачити результати роботи програми користувача;

User screen (Alt+F5) – відкрити «екран користувача», тобто перегляд результатів вашої роботи на комп'ютері поза середовищем, на повний екран.

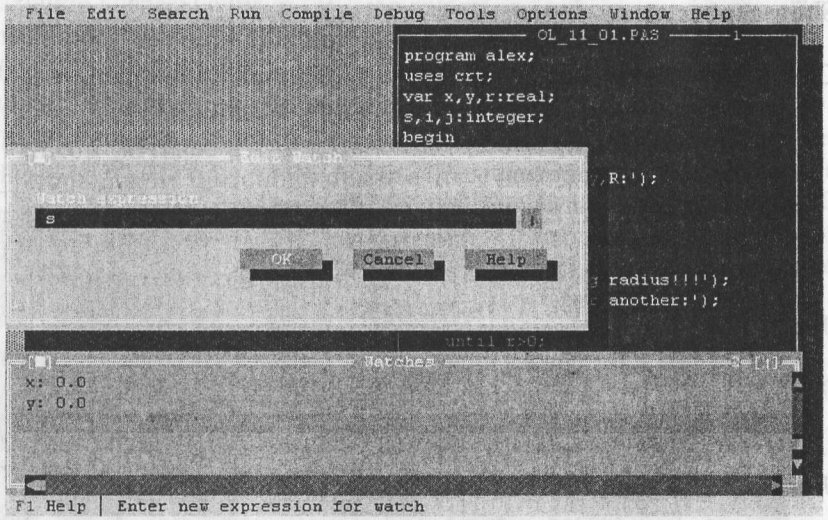
Зауваження. В інтегрованому середовищі Turbo Pascal 6.0 **Output** і **User screen** знаходяться в меню **Window**.

Виконуючи програму, ви змогли побачити результат її виконання. Однак, після другого натиснення на клавішу **ENTER** на екрані монітора знову з'явилося вікно з текстом програми. Для повторного перегляду результатів виконання програми немає потреби знову запускати її на виконання. Скористайтеся комбінацією клавіш **Alt+F5** і ви знову побачите результати

виконання програми. Якщо у вас є необхідність постійного контролю результатів виконання вашої програми, можна відкрити вікно користувача за допомогою опції **Output**. У цьому випадку на екрані монітора буде розташовано два вікна: одне з текстом вашої програми, друге – з результатами її виконання.

Процес пошуку помилок називають «дебагом», налагоджувачем. У перекладі з англійської bug означає комаху, жука. І дійсно, логічні помилки подекуди бувають так глибоко сховані, що на їх аналіз та пошук витрачається багато часу. Виконувати складні покрокові дії без допомоги комп'ютера дуже важко. І ось тут на допомогу програмісту приходить саме інтегроване середовище, де передбачене покрокове виконання всієї програми або її фрагмента.

Watch – відкрити нове вікно для перегляду поточних значень вказаних змінних (мал.12).



Мал.12

Для надання імені змінній необхідно у вікні **Watch** натиснути на клавішу **INSERT** або двічі натиснути ліву кнопку мишки. Результатом цієї дії буде поява на екрані ще одного вікна, де й вводять ім'я необхідної змінної (якщо ця змінна є іменем масиву, то у вікні **Watch** будемо спостерігати за зміною значень всіх елементів масиву одночасно). В інтегрованому середовищі Turbo Pascal 6.0 меню **Watch** розташоване в меню **Window**.

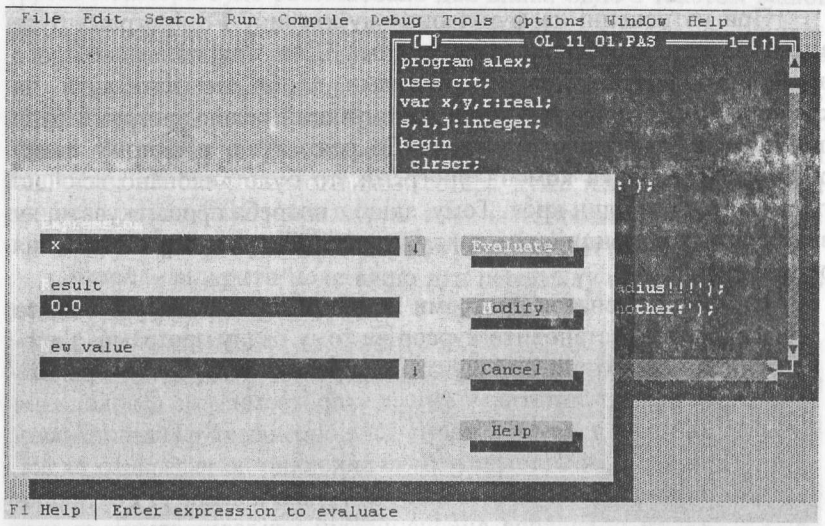
Звичайно, що переглядати поточну зміну значень величин має смисл лише при покроковому виконанні програми. В інтегрованому середовищі передбачена і така можливість. Розглянемо декілька її варіантів.

При натисненні на функціональну клавішу **F8** програма буде виконуватись у покроковому режимі. Для виконання кожного наступного рядка в тексті програми необхідно натискати на клавішу **F8**. На кожному кроці виконуваний рядок програми буде помічатися блакитною смужкою. До речі, якщо в одному рядку розмістити декілька команд програми, то буде виконано всю цю групу команд за один крок. Тому, якщо є потреба прослідкувати, як виконуються команди в тілі циклу, то рекомендується розбити їх на окремі рядки.

1. Якщо початок програми не викликає ніяких сумнівів, то можна встановити курсор на тому рядку програми, з якого необхідно виконувати покроковий перегляд проміжних результатів. Запуск програми на виконання здійснюється при цьому за допомогою функціональної клавіші **F4**. Програму буде виконано до вказаного рядка, а далі, користуючись вже знайомою клавішею **F8**, продовжите покрокове виконання дій і відслідковування значень змінних у вікні **Watch**.
2. Якщо ще до завершення виконання програми ви знайшли помилку, припинити її подальше виконання можна за допомогою комбінації клавіш **Ctrl+F2**.
3. Якщо вас цікавить конкретний фрагмент у програмі і ви хочете «прокручувати» цикл, зупиняючись весь час на цьому місці, то його можна позначити «червоним рядком» за допомогою комбінації клавіш **Ctrl+F8**. Таких контрольних зупинок можна встановити в програмі декілька. Виконання програми між ними здійснюється за допомогою комбінації клавіш **Ctrl+F9** або покроково клавішею **F8**. Зняття контрольних точок здійснюється тією ж комбінацією клавіш (**Ctrl+F8**) в самому позначеному рядку.

Існує ще одна можливість для перегляду поточного значення будь-якої змінної у програмі (мал.13). Не відкриваючи вікна **Watch**, під час зупинки виконання програми ви можете скористатися комбінацією клавіш **Ctrl+F4**. При цьому відкриється нове вікно, де можна набрати ім'я змінної величини, значення якої потрібне на даний момент, і переглянути його у наступному рядку вікна.

Виконайте розглянуті можливості середовища, проконтролювавши значення змінної *name* перед запуском програми на виконання, протягом виконання та після її завершення.



Мал.13

Питання для самоконтролю:

1 *Що означає термін «інтерактивне інтегроване середовище»?*
2 *Які функції виконує редактор в середовищі програмування?*
3 *Які функції виконує транслятор в середовищі програмування?*
4 *Які функції виконує налагоджувач в середовищі програмування?*
5 *Які основні функції інтегрованого середовища Turbo Pascal 7.0?*
6 *Якими «гарячими клавішами» інтегрованого середовища зручно користуватися під час налагодження програми?*

11. Практична робота «Робота в середовищі програмування»

За наведеним сценарієм виконайте завдання по створенню та налагодженню програми в середовищі Turbo Pascal 7.0.

1) *Запустити середовище Turbo Pascal 7.0 за допомогою файла turbo.exe.*
2) *Відкрити нове вікно за допомогою елемента меню File-New.*
3) *Набрати такий текст програми:*

```
program square;
 var a, b, s : integer;
```



```
begin
  write ('Задайте ціле число, що є шириною прямокутника: ');
  readln (a);
  write ('Задайте ціле число, що є висотою прямокутника: ');
  readln (b);
  s:=a*b ;
  writeln ('Площа прямокутника зі сторонами ',a,' та ', b,' дорівнює ',s)
end.
```

4) *Зберегти текст набраної програми за допомогою елемента меню File-Save as….*
5) *Запустити на виконання програму з активного вікна за допомогою «гарячих клавіш» Ctrl+F9.*
6) *Виправити помилки, якщо вони будуть знайдені, і запустити програму на виконання ще раз.*
7) *Зберегти налагоджену програму за допомогою «гарячої клавіші» F2. Переглянути результати роботи програми за допомогою «гарячих клавіш» Alt+F5 і повернутися назад в середовище за допомогою клавіші ESC.*
8) *Відкрити вікно Watch та проконтролювати значення змінних a, b, s протягом покрокового виконання програми.*
9) *Створити exe-файл програми за допомогою «гарячих клавіш» Alt+F9 (в режимі Destination-Disk).*
10) *Закрити вікно з виконаною програмою за допомогою клавіш Alt+F3.*
11) *Завершити сеанс роботи в інтегрованому середовищі програмування Turbo Pascal 7.0 за допомогою «гарячих клавіш» Alt+X та запустити на виконання одержаний exe-файл програми з операційної системи або з її оболонки.*
12) *Скоректувати текст програми, наведеної у п.3 таким чином, щоб вона обчислювала площу квадрата, і виконати п.п.1-11.*

12. Мова програмування Паскаль

Особливості та призначення

У 1970 році з'явилося повідомлення про створення ще однієї мови програмування, названої на честь відомого вже нам математика, автора однієї з перших механічних обчислювальних машин Блеза Паскаля. Автором цієї мови програмування був Ніклаус Вірт, професор, директор Інституту інформатики Швейцарської вищої

політехнічної школи, лауреат Тьюринговської премії, автор численних та широко відомих праць в області програмування. Н. Вірт є автором ще таких мов програмування, як Ейлер, Модула, Модула-2. Окрім цього, ним запропонована методика покрокової розробки програм – від глобального до локального, від загального до часткового, – тобто занурення в алгоритм зверху вниз. Ця методика була визнана найсильнішою ідеологією програмування 70-х років.

Вірт вирішив взятися за створення нової мови програмування з методичної точки зору: дати своїм студентам інструмент для вивчення програмування як систематичної, логічної дисципліни, що базується на фундаментальних поняттях.

У своїй роботі «Систематичне програмування» Ніклаус Вірт пише: «...мова, якою студента навчають висловлювати свої думки, здійснює глибокий вплив на його навички мислення та винахідницькі здібності...».

Мова програмування Паскаль, на відміну від усіх попередніх популярних мов, дала програмісту в руки прості конструкції команд, що виглядають дуже природно. Тобто їх зміст співпадає з тим, чого інтуїтивно чекає користувач. Набір команд мови Паскаль, наприклад, у порівнянні з мовою програмування Basic, зовсім невеликий і складає її базову частину. Усі додаткові можливості Паскаля дають змогу програмісту користуватися типами змінних, що властиві лише цій мові програмування, а також створювати свої власні типи.

Дуже швидко мова Паскаль набула великої популярності серед програмістів і була реалізована для різних типів комп'ютерів.

Ми розглядатимемо інтегроване середовище програмування Turbo Pascal 7.0 і будемо відзначати в разі необхідності його відмінності від інтегрованого середовища програмування Turbo Pascal 6.0, хоча усі команди та приклади, що будуть наведені далі, можуть бути використані і в усіх попередніх версіях.

Ви зможете належним чином оцінити всю красу та привабливість мови програмування Паскаль і після її вивчення у вас обов'язково з'явиться потреба в поглибленні своїх програмістських навичок.

Алфавіт мови програмування

Опис будь-якої мови, в тому числі і алгоритмічної, містить в собі алфавіт, синтаксис і семантику. Уточнимо ці поняття.

Алфавіт мови програмування – це набір символів, які можуть бути використані при складанні програми.

Клавіатура комп'ютера містить всі символи алфавітів, що використовуються в будь-яких мовах програмування. Знайомство з мовою програмування починається саме з алфавіту: користувачу необхідно знати, які символи «розуміє» дана мова програмування. Ясно, що для того, аби без помилок написати програму, необхідно використовувати лише знаки заданого алфавіту.

Наступним кроком у знайомстві з мовою програмування є вивчення правил запису операторів та описів, які припустимі в даній мові.

Синтаксис визначає правила побудови із символів алфавіту спеціальних конструкцій, за допомогою яких можна реалізувати алгоритми розв'язування задач.

Але далеко не завжди синтаксично правильно написана програма повинна видати очікуваний результат. Як і в будь-якій розмовній мові правильно записані з точки зору синтаксису речення повинні мати зміст.

Семантикою називають систему правил тлумачення конструкцій мови програмування.

Синтаксичні помилки визначаються компілятором системи програмування, семантичні – системою тестів, підібраних самим користувачем. Навчити читача робити якомога менше синтаксичних, а головне семантичних помилок і є першочерговим завданням даного посібника.

А зараз детальніше ознайомимося з алфавітом мови програмування Паскаль та зарезервованими словами. Його складають:

символи, що використовуються для утворення імен змінних.

До них належать:

- латинські малі та великі букви;
- арабські цифри від 0 до 9, які в іменах змінних можуть використовуватися лише з другої позиції;
- символ підкреслення;

символи-розділювачі. До них належать

- символ пробілу, основне призначення якого – це розділення ключових слів та імен;
- керуючі символи, які можуть використовуватися при описові рядкових і символьних констант;

спеціальні символи, що виконують певні функції при побудові різноманітних конструкцій мови:

+ - * / { } [ ] ( ) < > . , ' : ; ^ @ # $

складені символи, які сприймаються компілятором як єдине ціле:

<= => := (* *) (. .) ..

«невикористовувані символи» – це символи, які не відносяться до п. п. 1-4, але які можна використовувати в текстах коментарів та в якості значень рядкових та символьних констант;

зарезервовані слова, тобто ключові слова, які входять до словника мови програмування і які не можна використовувати в якості імен змінних. До таких ключових слів відносяться службові слова для запису операторів (**if**, **then**, **else**, **while**, **repeat** тощо), визначення розділів Паскаль-програми (**const**, **type**, **var** тощо), інші ключові слова (**begin**, **end**). Для зручності в даному посібнику у наведених прикладах програм всі зарезервовані слова виділені жирним шрифтом.

До алфавіту мови програмування Паскаль вам доведеться звертатися щоразу протягом роботи з даним посібником. Саме під час практичного засвоєння цих нових понять можна отримати позитивний результат.

Основні поняття мови

До основних понять мови програмування належить: ***оператори, ідентифікатори, числа, символи, рядки***.

В основі будь-якої мови програмування лежить поняття оператора.

Оператор – це самостійна одиниця мови, яка описує зміст відповідного етапу алгоритмічного процесу. Алгоритм задається послідовністю операторів.

Оператор є тим самим приписом, з послідовності яких складається алгоритм. В мові програмування визначається фіксована кількість типів операторів, кожний з яких призначений виконувати певний набір дій. При складанні програми дозволяється використовувати лише ці оператори. Запис алгоритму мовою програмування є послідовністю операторів. Оператори в свою чергу використовують більш дрібні конструкції.

В переважній більшості програм використовуються змінні величини, а для їх позначення – імена або ідентифікатори.

Ідентифікатори або імена використовуються для позначення змінних величин.

Зауважимо, що у Паскалі поняття ідентифікатора значно ширше. Ідентифікаторами позначаються також і константи, і типи даних, і процедури та функції. Про можливості надання цим елементам програми ідентифікаторів будемо знайомитися поступово.

Величина, яка не є змінною, відповідно називається константою. В програмі константи представляються числами, символами, текстами.

Представленню чисел слід приділити особливу увагу, а саме слід відповісти на таке слушне запитання: яким чином числа зберігаються у пам'яті комп'ютера?

Що стосується цілих чисел, то немає ніяких проблем: цілі числа задаються користувачем і виводяться програмою у традиційному вигляді.

Для подання дійсних чисел у Паскалі та й в інших мовах програмування, існує дві форми.

Природна форма дійсного числа – це традиційний запис дійсного числа.

Необхідно лише пам'ятати, що в мовах програмування, на відміну від математичного запису, ціла частина дійсного числа відділяється від дробової крапкою, а не комою.

Наведемо декілька прикладів:

10.123, 1.0123, 1012.3, 0.0010123 .

Всі ці числа різні за значенням, але значуща частина, або мантиса, у них однакова.

Показникова форма дійсного числа – це представлення його у вигляді мантиси і порядку із вказівкою їх знаків.

Не завжди числа, з якими мають справу при програмуванні, виглядатимуть так зручно. Подекуди це будуть дуже великі або дуже маленькі числа, для яких задавати зайві нулі не зовсім зручно.

Розглянемо приклад малого $\mathbf{0.00000000010123 = 1.0123*10^{-10}}$ або дуже великого числа $\mathbf{10123000000000000000=1.0123*10^{20}}$.

Отже, у показниковій формі числа можна виділити такі основні характеристики: знак числа, мантису числа, знак порядку та порядок числа. Усі решта елементів запису числа у показниковій формі повторюються – знак множення, основа степеня (10). Це

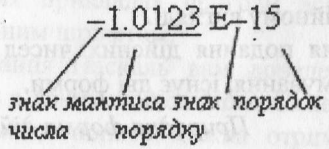
означає, що запам'ятовувати їх у пам'яті комп'ютера не має ніякого сенсу. І дійсно, саме зазначені чотири характеристики дійсного числа зберігаються у пам'яті.

Ви, мабуть, чекаєте відповіді на запитання: це ж не запис числа, а подання його у вигляді арифметичних дій множення та піднесення до степеня? Щоб уникнути цих арифметичних дій, дійсне число у показниковій формі в Паскалі подається таким чином: **1.0123E-10** .

У цьому записові символ E означає основу степеня 10, і комп'ютер розпізнає цей запис, як форму зображення дійсного числа. Характерно, що додатні знаки числа і мантиси можна не вказувати, вони будуть встановлені автоматично.

Визначимо схематично основні характеристики дійсного числа (мал.14).

Щодо символів, які можуть використовуватись в якості констант, то ми вже визначили їх в алфавіті мови Паскаль. Рядки – це послідовність символів. І символи, і рядки можуть використовуватись в Паскаль-програмі як константи, і як коментарі. Як саме це робиться, зараз говорити передчасно. Опановуючи цей посібник, ми з вами будемо зустрічатися з основними поняттями мови програмування на кожному кроці.



Мал. 14

Структура Паскаль-програми

Мову програмування можна собі уявити як деякий засіб спілкування з комп'ютером. Тому, як і у будь-якій іншій мові спілкування, тут є певні домовленості у поданні своїх висловів: своя абетка та синтаксичні правила.

Синтаксично Паскаль-програма складається з окремих «речень». Такими «реченнями» можуть бути деякі описи для самої програми або команди, з яких складається даний алгоритм.

Кожне «речення» Паскаль-програми повинно закінчуватися символом «;».

Послідовність представлення алгоритму у вигляді Паскаль-програми має певну закономірність.

Визначимо її так:

program <ім'я програми> ;
uses <розділ опису бібліотек (модулів), що підключаються>;
label <розділ опису міток>;
const <розділ опису констант>;
type <розділ опису типів>;
var <розділ опису змінних>;
procedure або **function** <розділ опису процедур і функцій>;
<розділ операторів>

Ви, мабуть, звернули увагу, що службові слова Паскаль-програми, як і було зазначено раніше, виділені жирним шрифтом. Надалі службовими словами вважатимемо ключові слова, з яких складаються оператори Паскаля, і назви його стандартних процедур та функцій. Окрім того, вся програма розбивається на певні розділи. Бажано зберігати таку послідовність використання розділів, хоча, набуваючи досвіду, ви побачите, що деякі з них можна міняти місцями. Однак треба враховувати, що розділ опису модулів повинен обов'язково йти першим, а розділом операторів завершується кожна програма. Розділ типів обов'язково повинен передувати розділу змінних, а розділ міток та розділ констант бажано розміщувати перед розділом типів. Першим рядком програми зі службовим словом **program** можна знехтувати.

Умовно цю структуру можна поділити на дві частини – описову частину та виконувану.

*Описова частина (**program, uses, label, const, type, var**) містить інформацію про можливості, які можна використовувати у програмі, які будуть константи, мітки, якими змінними користуватися.*

Саме за змістом цієї описової частини всім вказаним змінним відводиться місце у пам'яті комп'ютера залежно від їх типів і послідовності, в якій вони зазначені в програмі.

*Виконувана частина містить опис процедур, функцій та основний блок програми, який ще називається тілом програми і розташований між службовими словами **begin** та **end**.*

Завершується Паскаль-програма завжди символом « . » .

Як приклад наведемо програму, за допомогою якої можна обчислити площу круга, вказавши будь-який радіус.

```
program circle;
 var r, s : real;
begin
   write ('Задайте радіус круга: ');
   readln (r);
   s:=pi*r*r ;
   writeln ('Площа круга з радіусом ',r,' дорівнює ',s)
end.
```

Зверніть увагу на принцип вкладеності, за допомогою якого досягається максимальна «читабельність» вашої програми. Вміст розділу операторів зміщений трохи вправо відносно службових слів **begin** та **end.**. Аналогічно записана і описова частина Паскаль-програми. Кожне «речення» програми записується з нового рядка. Оператори, які є вмістом інших, зміщуються відносно них так само вправо. Це дозволяє краще розібратися у складних програмах, знайти в них помилки не тільки самому автору, але й сторонньому користувачу.

Питання для самоконтролю:

1 *Які особливості та призначення мови програмування Паскаль?*
2 *Хто є розробником мови Паскаль?*
3 *Що таке алфавіт мови програмування?*
4 *Що таке синтаксис мови програмування?*
5 *Що таке семантика мови програмування?*
6 *Назвіть символи, які входять до алфавіту мови Паскаль.*
7 *Назвіть основні поняття мови програмування.*
8 *Що називається оператором?*
9 *Що таке ідентифікатор?*
10 *У якому вигляді зберігаються у пам'яті комп'ютера дійсні числа?*
11 *Які є форми представлення дійсних чисел в мовах програмування?*
12 *Як символи та рядки можуть використовуватися в Паскаль-програмі?*
13 *Назвіть основні розділи Паскаль-програми.*
14 *На які частини можна поділити Паскаль-програму? Яке їх призначення?*

Лінійні алгоритми

13. Величини

Поняття величини. Основні характеристики величин

Алгоритми, так само як і програми, в першу чергу мають справу з ***величинами***. Енциклопедичний словник тлумачить поняття величини як узагальнення конкретних понять: довжини, площі, ваги тощо. Вибравши одну з величин даного роду за

одиницю виміру, можна виразити числом відношення будь-якої іншої величини цього ж роду до одиниці виміру. Тобто це поняття носить кількісний, порівняльний характер.

З точки зору алгоритмізації в якості величин та їх сукупності виступають дані, що обробляються цими алгоритмами. Якщо ж розглядати алгоритм на рівні його виконання на комп'ютері, то в цьому випадку значення величини трактується як інформація, якою оперує програма, а тому величині повинно бути відведено місце в пам'яті. Йдучи ще далі в наших міркуваннях, слід визнати, що сама програма, з точки зору комп'ютера, також є послідовністю даних, тобто програмних елементів (команд, операторів), кожній з яких при введенні відводиться місце в пам'яті.

Отже, зупинимося на тому, що саме через величини передаються усі значення, вся інформація, виконуються різні обчислення.

Величини можна поділити на ***сталі*** та ***змінні***.

Сталі – це такі величини, які не змінюють свого значення під час виконання всього алгоритму.

Наведемо декілька прикладів сталих величин:

10, –123.45, 'A', 'ліцей', 'моя програма'.

Що незвичного у цих прикладах? Перше число – 10 – є цілим, і його запис не викликає ніяких запитань. А от у другому числі, що є дійсним (–123.45), ви, мабуть, звернули увагу на те, що ціла частина числа відділяється від дробової знаком «.», а не «,», як ви звикли у математиці.

Слід нагадати, що у всіх мовах програмування у записові дійсних чисел ціла частина числа відділяється від дробової знаком «.».

Наступні три приклади сталих величин є дещо специфічними. Стала величина 'A' – це знайомий нам символ «A», який використовується як складова частина деякого тексту. Два останніх приклади є сталими величинами-текстами (рядками), які використовуються у програмах як коментарі при введенні початкових даних, виведенні результатів або текст.

На відміну від значень сталих величин, які ми ніяким чином під час алгоритму змінити не можемо, змінні величини характерні тим, що їх значення змінюються.

Змінними називаються такі величини, які можуть змінювати своє значення під час виконання алгоритму.

Як же позначити ці величини, якщо їх значення можуть змінюватися протягом виконання програми?

У математиці та фізиці такі величини одержують якесь ім'я. Наприклад, S – площа, V – об'єм, F – сила, m – маса тіла, U – напруга електричного струму.

У програмуванні є певні правила для найменування змінних величин.

Іменем (ідентифікатором) змінної величини може бути будь-яка послідовність латинських літер (a–z та A–Z), цифр (0–9) та знаків підкреслення, що починається з літери.

Характерно, що у Паскаль-програмах не розрізняються великі та маленькі літери. Тобто ідентифікатори *MyResult* та *myresult* вважаються однаковими. Але все ж таки слід рекомендувати застосовувати ідентифікатори змінних переважно за їх змістовним призначенням для того, щоб ваша програма була «читабельною». І саме цього ми будемо дотримуватися у прикладах, що зустрічатимуться у даному посібнику.

На кількість символів, з яких складаються імена змінних, накладається обмеження: ідентифікатори не повинні містити більше, ніж 63 символи. Можна використовувати і довші ідентифікатори, але всі символи після 63-го ігноруються. А оскільки кожне ім'я змінної повинно бути в даній програмі ***унікальним***, то можна припуститися помилки, якщо ці ідентифікатори будуть відмінні у символах після 63-го.

Позначивши змінну ідентифікатором, ще потрібно визначити, яких саме значень може набувати ця величина.

З поняттям величини завжди пов'язують поняття типу: ціла, дійсна, логічна, символьна або рядкова. Ми вже знаємо, що на рівні виконання програми кожній величині відводиться відповідне місце в пам'яті комп'ютера, розмір якого залежить саме від типу величини. Яке місце в пам'яті займають змінні різних типів, з якими працює Паскаль, розглянемо далі.

Наявність різноманітних типів величин у Паскалі можна пояснити такими двома моментами:

- економія пам'яті комп'ютера при виконанні програм;
- перетворення типів величин під час виконання операцій, застосованих до них.

Перше твердження не повинно викликати ніяких непорозумінь. Особливо гостро це питання постане, якщо матимемо справу зі створенням програм значних розмірів. А от стосовно другого, то на ньому варто зупинитись детальніше. Кожна операція

у Паскалі вимагає величин певного типу і видає результат так само певного типу. Саме тому при знайомстві з операціями Паскаля ми будемо звертати увагу на типи величин, якими вони оперують, та на тип результату, який одержується.

Якщо сталі величини не вимагають визначення їх типу, оскільки вони задаються явним чином, то для змінних величин обов'язково необхідно визначати тип.

І ще одна дуже суттєва деталь: у кожний момент часу при виконанні програми змінна величина повинна мати якесь значення!

Отже, про сталу величину за її записом зразу можна сказати все: яке її значення, якого вона типу.

*Змінна величина повинна характеризуватися такими ознаками: **ім'ям, типом та значенням**.*

Стандартні типи даних

У Паскалі закладені найпоширеніші типи змінних величин, які називаються стандартними, тобто такі, які зрозумілі компілятору без додаткових пояснень.

Типи змінних, що набувають цілих значень, позначаються службовим словом **Integer**. Існують певні межі для значень величин цього типу: **–32768 .. 32767** .

У пам'яті комп'ютера такі значення займають 2 байти.

У Паскалі існує можливість використання ще декількох різновидів цілих типів, які мають інші діапазони своїх значень.

Подамо всі типи цілих чисел у табличному вигляді (табл.2).

Якщо вам наперед відомо, що деяка змінна може набувати досить невеликих значень протягом виконання алгоритму, то її можна описати типом **Shortint**, тобто коротке ціле. У випадку, якщо вам навпаки потрібен дуже великий розбіг значень цілої змінної, можна скористатися типом **Longint**. Бувають такі програми, що працюють тільки з цілими додатними числами. І для такого випадку у Паскалі є відповідні типи – тип **Byte** та тип **Word**.

Таблиця 2

Тип	Діапазон	Формат	Розмір в байтах
Integer	-32768 .. 32767	Знаковий	2
Shortint	-128 .. 127	Знаковий	1

Longint	-2147483648 .. 2147483647	Знаковий	4
Byte	0 .. 255	Беззнаковий	1
Word	0 .. 65535	Беззнаковий	2

Дійсні змінні у Паскалі позначаються типом **Real**. Їх значення займають у пам'яті комп'ютера ***6 байтів***, дають 11–12 значущих цифр числа та визначаються інтервалом зміни точності значень:

$$2.9*10^{-39} .. 1.7*10^{38},$$

тобто числа, менші лівої межі вважаються рівними 0, а більші правої набувають значення $1.7*10^{38}$.

Окрім основного типу дійсних чисел, існують його різновиди, які відрізняються кількістю значущих цифр числа, кількістю байтів, які воно займає у пам'яті комп'ютера, та інтервалом точності значень. Для порівняння наведемо їх у вигляді таблиці (табл.3):

Таблиця 3

Тип	Діапазон	Число значущих цифр	Розмір у байтах
Real	$2.9*10^{-39}..1.7*10^{38}$	11–12	6
Single	$1.5*10^{-45}..3.4*10^{38}$ї	7–8	4
Double	$5.0*10^{-324}..1.7*10^{308}$	15–16	8

Зауваження. Дії над стандартними типами з одинарною точністю **Single** та з подвійною точністю **Single** можуть виконуватися лише за наявності числового сопроцесора. Для того, щоб скористатися його можливостями, необхідно у меню **Options-Compiler** встановити [X] 8087/80287 у полі **Numeric processing**, або комбінацією клавіш **CTRL+O+O** у самому першому рядку текста програми встановити стандартні директиви компіляції програми.

Знайомлячись з прикладами сталих величин, ви звернули увагу на те, що величини, якими оперує Паскаль, можуть бути не тільки числовими.

Тип **Char** (від англійського слова *character*) описує символьні змінні, тобто такі, значенням яких може бути будь-який символ із таблиці ASCII-кодів. Коди цих символів займають у пам'яті ***1 байт*** і їх можлива кількість складає 256 (2^8=256).

Змінні, значеннями яких є послідовність символів, визначаються типом **String**, що перекладається як «рядок». Кількість символів у таких рядках обмежена числом 255. Об'єм пам'яті, який

займають дані цього типу, складає ***256 байт***. Згодом у нас буде можливість детальніше познайомитися і попрацювати із змінними цього типу, а поки що наведеної інформації цілком достатньо.

Завершуємо ми цей невеличкий огляд стандартних типів змінних ще одним типом, що має назву **Boolean** – логічний тип. Змінні цього типу можуть набувати лише двох значень – **False** та **True** (істинно та хибно). Розмір пам'яті для значень цього типу складає всього ***1 байт***.

Питання для самоконтролю:

1 *Як ви розумієте поняття «величина»?*
2 *Як трактується поняття величини з точки зору її відображення на пам'ять комп'ютера?*
3 *Що називається сталою величиною? Наведіть приклади.*
4 *Яка специфіка запису символьних та рядкових сталих величин?*
5 *Що називається змінною величиною?*
6 *Якими ознаками характеризується змінна величина?*
7 *Які стандартні типи величин використовуються у Паскалі?*
8 *Які різні типи цілих величин визначені у Паскалі? Чим вони відрізняються?*

14. Введення та виведення інформації

Вказівки введення та виведення інформації

Для створення простої програми необхідно познайомитися з можливістю введення значень аргументів та виведення результатів виконання програми.

У Паскалі для цієї мети існують вже готові, а значить стандартні, процедури. Подекуди ці процедури називають операторами. Але кінець кінцем, важлива суть, а не форма.

Загальний вигляд процедури введення інформації:
read *(<список елементів введення>) або*
readln *(<список елементів введення >).*

Елементами списку введення можуть бути один або декілька ідентифікаторів, записаних через кому. Наприклад:

read (a,b,c);
read (alfa);
readln (m,n).

При виконанні процедур **read** та **readln** програма переходить у стан очікування введення даних. Причому процедури **read** та **readln** при введенні інформації з клавіатури мало чим відрізняються одна від

одної. При виконанні програми можна помітити те, що після введення інформації за допомогою процедури **readln** курсор переходить на початок нового рядка, а після роботи процедури **read** – ні.

Зауваження:

- якщо передбачається введення зразу декількох значень, то їх можна вводити в одному рядку, відділяючи одне від одного символом «пробіл», або в окремих рядках, завершуючи введення кожного значення клавішею ENTER;
- процедури не завершать своєї роботи, поки не будуть введені значення для всіх змінних;
- змінні одержують свої значення послідовно в процесі введення даних (перше значення – першій змінній, друге – другій і т.д.);
- тип значень, що вводяться, повинен відповідати тому типові, яким описана дана змінна, інакше при введенні інформації буде видано повідомлення про помилку.

Як ви думаєте, яке значення одержить змінна a після виконання процедури read(a,a,a), якщо буде набраний такий рядок значень: 10 -5 1 ? Ви не помилились: змінна a одержить значення 1 (з третього разу)!

Загальний вигляд процедури виведення інформації:
***write** (<список елементів виведення>) або*
***writeln** (<список елементів виведення>).*

Список елементів виведення інформації набагато ширший. В ньому можуть використовуватися:

- ідентифікатори, значення яких будуть виведені на екран;
- вирази, значення яких спочатку будуть обчислені, а потім виведені на екран монітора;
- сталі величини (числові, символьні, рядкові).

Різниця між процедурами **write** та **writeln** така: процедура **write** після виведення інформації лишає курсор на першій вільній позиції, а процедура **writeln** – переводить курсор на початок нового рядка. Приклади:

write ('Задайте значення трьох цілих чисел');
write (a+b,' – така одержана сума двох чисел');
write ('a=',a,' b=',b);
writeln ('a=',10).

Ще раз зверніть увагу на те, що всі коментарі є рядковими сталими величинами і їх необхідно брати в апострофи. Причому, апострофів у списку елементів виведення повинна бути парна кількість (третій приклад). Якщо текст коментаря передбачає використання апострофів, то замість одного апострофа в тексті необхідно поставити два ('комп'ютер').

Оскільки елементами процедур введення інформації не можуть бути тексти, необхідно обов'язково використовувати перед ними процедури виведення інформації з відповідними текстами для того, щоб ваша програма не була «німою», щоб вам було зрозуміло, скільки даних та які саме необхідно задати для її виконання.

У четвертому прикладі на екран монітора завжди буде виводитися один і той же вираз, незалежно від того, яке значення змінна a буде одержувати у програмі: **a=10.**

А як ви думаєте, що виведе ваша програма у другому прикладі, якщо величинам *a* та *b* не будуть задані значення? Арифметичний вираз буде обчислено, але в якості значень *a* та *b* буде взяте з пам'яті комп'ютера так зване «сміття», тобто те, що знаходилося там в той момент, коли частина пам'яті розподілялася для цих змінних. Це можуть бути залишки інформації попередньої програми, яка знаходилися на цьому місці, або випадково нулі, якщо ви тільки увімкнули комп'ютер, і в цю область оперативної пам'яті ще ніяка інформація не була занесена. Згадайте це зауваження, якщо зустрінетеся з ситуацією, коли програма при кожному запускові видає непередбачувані результати!

І ще одна можливість процедур виведення інформації, яку можна використовувати для одержання результатів у зручному, не показниковому вигляді. Це форматування значень змінних.

*Вигляд форматування: <ім'я змінної>: **n : m**,*
*де **n** – загальна кількість символів, що відводяться під значення змінної, а **m** – кількість знаків дробової частини.*

У загальну кількість символів входить і знак числа, і крапка, що відділяє дробову частину числа від цілої. Цей тип форматування, звісно, задається тільки для дійсних чисел. Для форматування цілих чисел параметр *m* зайвий.

Для більшої ясності наведемо кілька прикладів з поясненнями.

write('сума=',S:10:6);

write('кількість елементів:',n:5);
writeln('a=',a:0:3,' b=',b).

У першому прикладі при S=–101.697 буде виведена на екран
сума=–101.697 ,
у другому при n=57 –
кількість елементів:_ _ _57 ,
у третьому при a=10.2458 та b=–132.879 –
a=10.246 b=–1.32879E+2 .

З цих прикладів можна зробити такий висновок. При форматуванні у дробовій частині при зайвих зарезервованих позиціях у кінці числа дописуються нулі, а при їх нестачі відбувається округлення останньої можливої цифри. У цілій частині надрукується стільки цифр, скільки необхідно для даного числа, навіть якщо є нестача позицій, або ж перед цілою частиною буде виведена необхідна кількість пробілів, якщо вказано позицій більше, ніж потрібно. При відсутності форматування значення дійсних чисел будуть надруковані у показниковій формі.

Скориставшись цією інформацією, ви швидко оціните зручність форматування вихідних даних.

Використання процедур та функцій модуля CRT

Монітори персональних комп'ютерів можуть працювати у двох режимах – текстовому та графічному. Після запуску програми на виконання автоматично задається текстовий режим роботи, при якому на екран може виводитися 80 символів у 25 рядках. Ознайомлення з графічним режимом роботи монітора чекає нас попереду. Але й для текстового режиму існує багато цікавих додаткових можливостей. Усі вони реалізовані у вигляді процедур та функцій, розміщених у модулі **CRT**. У Паскалі під модулями розуміють бібліотеки, в яких розміщені процедури та функції, що виконують роль допоміжних алгоритмів. Вони знаходяться в окремих файлах і підключаються до вашої програми тільки тоді, коли їх імена вказані у розділі **Uses**. Це дозволяє не «перевантажувати» Паскаль-програму зайвою інформацією.

Найбільш використовувані модулі (серед них і модуль **CRT**) поміщені у файл turbo.tpl (tpl – Turbo Pascal Library). Тому перед використанням можливостей цього модуля перевірте наявність файла turbo.tpl на диску.

Для підключення модуля **CRT** до програми необхідно до неї включити такий рядок:

```
program my_prog;
uses crt;
begin
.....
end.
```

Наведемо список основних процедур модуля **CRT**.

ClrScr –очищення екрану монітора;

GoToXY(x,y) –встановлення курсора в позицію x рядка y (1<=x<=80, 1<=y<=25);

TextColor(color) –встановлення кольору тексту;

TextBackGround(color) – встановлення кольору фону символів;

Delay(Ms) –затримка виконання програми на вказаний у мікросекундах термін часу (спробуйте Delay(200));

Sound(Hz) –увімкнення звукового сигналу вказаної частоти (у герцах);

NoSound –відключення звукового сигналу.

Залишилися невідомими значення параметрів (аргументів) процедур для роботи з кольорами. Наведемо їх значення у вигляді таблиці (табл.4):

Таблиця 4

0	**Black**	чорний
1	**Blue**	синій
2	**Green**	зелений
3	**Cyan**	блакитний
4	**Red**	червоний
5	**Magenta**	фіолетовий
6	**Brown**	коричневий
7	**LightGrey**	яскраво-сірий
8	**DarkGrey**	темно-сірий
9	**LightBlue**	яскраво-синій
10	**LightGreen**	яскраво-зелений
11	**LightCyan**	яскраво-блакитний
12	**LightRed**	рожевий
13	**LightMagenta**	бузковий
14	**Yellow**	жовтий
15	**White**	білий
128	**Blink**	блимання символа

При вказанні необхідного кольору можна користуватися або числовими значеннями, або ж їх мнемонічними альтернативними позначеннями – це одне й те ж саме. Паскаль все зрозуміє!

Процедури дуже схожі на оператори Паскаля, бо записуються окремими «реченнями» та завершуються символом «;». Серед них є такі, яким необхідно передати для виконання аргументи, а інші виконуються без усякої додаткової інформації.

Вам, напевно, доводилося писати заяви? Давайте розберемо алгоритм цієї задачі за домовленості, що ми заповнюємо уявний бланк заяви при оформленні на роботу.

Задати ім'я директора організації, якому подається заява.

Задати ім'я заявника.

Вказати дату подачі заяви.

Внести дані, зазначені у п.п.1-3, до тексту заяви.

Програма, що відповідає наведеному алгоритму та використовує можливості виведення інформації на екран монітора, передбачає зображення на екрані монітора тексту заяви за всіма вимогами щодо оформлення цього документу. Переконайтеся в цьому, набравши її текст і запустивши на виконання.

```
program statement;
 uses crt;
 var name_direct, my_name, data: string;
begin
   writeln ('Введіть ім''я директора організації');
   readln (name_direct);
   writeln ('Введіть своє ім'я');
   readln (my_name);
   writeln ('Введіть дату подачі заяви');
   readln (data);
   ClrScr;
   TextColor(Red);
   Sound(200);
   GoToXY(50,1);
   write ('Директору ', name_direct);
   GoToXY(50,2);
   write (my_name);
   GoToXY(35,6);
   write ('З А Я В А ');
   GoToXY(5,8);
```



```
   write('Прошу прийняти мене, ',my_name,' на роботу.');
   GoToXY(5,10);
   write (data);
   NoSound;
   Delay(500)
end.
```

Зверніть увагу на те, що значення змінних величин типу **string** треба задавати лише процедурами **readln**, а після використання процедур **GoToXY** йдуть процедури **write**, адже після встановлення курсора в потрібне місце екрана процедура **writeln** перемістить його на початок наступного рядка.

А тепер розглянемо дві цікаві функції модуля **CRT**.

KeyPressed – повертає значення **true**, якщо була натиснена будь-яка клавіша, та **false** – у протилежному випадку;

ReadKey – повертає значення символу, що відповідає натисненій клавіші.

Ви вже помітили, чим відрізняються функції від процедур? Процедури зразу виконують передбачені в них дії, а функції одержують певні значення, з якими далі необхідно виконати деякі дії. І ми зараз цьому навчимося!

Перш за все розберемося, як відбувається введення вами інформації з клавіатури. Клавіатура, як і інші периферійні пристрої, має свій буфер, тобто виділену частину оперативної пам'яті. Саме в ньому послідовно «збираються» коди тих клавіш, які ви натискаєте. Робота стандартних процедур **read** та **readln** організована таким чином, що тільки при натисканні клавіші **ENTER** всі накопичені в буфері дані певним чином перетворюються і присвоюються черговій змінній. Після цього буфер клавіатури спорожнюється і готовий для одержання нових даних.

Повернемося до наших функцій. Функція **ReadKey** читає з буфера клавіатури код натиснутої клавіші, що знаходиться у буфері першою. Якщо перед виконанням цієї функції буфер був порожнім, то програма переходить у режим очікування, поки у буфер щось не буде занесене. У цьому випадку після виконання функції **ReadKey** буфер знову стає порожнім. Якщо ж при виконанні функції **ReadKey** буфер не був порожнім, то він спорожниться тільки тоді, коли функцію виконають стільки разів, скільки кодів клавіш є у

буфері. Саме цією особливістю можна скористатися, щоб зробити затримку виконання вашої програми до натискання будь-якої клавіші. Цією особливістю дуже зручно користуватися наприкінці програми, щоб мати можливість проаналізувати одержані результати, перш ніж середовище Turbo Pascal перейде у режим роботи з текстом вашої програми.

```
program my_prog;
 uses crt;
 var k: char;
begin
   { тіло програми }
   k:=ReadKey;
end.
```

Затримку виконання програми можна зробити і використавши процедуру **readln** без аргументів:

```
program my_prog;
 begin
  { тіло програми }
  readln;
 end.
```

Особливість використання такої «затримки» полягає в тому, що процедура **readln** реагує лише на натискання клавіші «Enter».

І ще один, третій варіант організації «затримки» роботи вашої програми. Це використання функції **KeyPressed**. Для пояснення нам доведеться забігти трошки наперед. Справа в тім, що функція **KeyPressed** нічого з буфера не вичитує, а лише перевіряє, чи там щось є. Скориставшись цією особливістю, запишемо оператор:

```
repeat until KeyPressed.
```

Прочитати цей запис можна таким чином: «повторювати, поки не буде натиснена якась клавіша». Поки буфер буде порожній, функція **KeyPressed** повертатиме значення **false** і ми будемо «висіти» на цьому фрагменті програми. Але як тільки буде натиснена будь-яка клавіша, функція **KeyPressed** поверне значення **true** і зазначену дію буде завершено.

Отже, варіант використання функції **KeyPressed**:

```
program my_prog;
 uses crt;
 begin
```



```
   { тіло програми }
   repeat until KeyPressed;
 end.
```

До речі, інформація у буфері після використання функції **KeyPressed** залишиться, тобто ця функція лише перевіряє наявність інформації в буфері клавіатури. Це означає, що всередині програми такий спосіб «затримки» треба використовувати дуже обережно і лише тоді, коли ви переконані, що перед цією операцією буфер клавіатури був порожнім. Може, колись вам стане у пригоді порада, яким чином спорожнити буфер клавіатури. Для цього можна використати такий фрагмент програми:

```
program empty;
 uses crt;
 var k:    char;
 begin
   { тіло програми }
   while KeyPressed do
         k:=ReadKey;
   { продовження тіла програми }
 end.
```

На останок розглянемо алгоритм обчислення середнього арифметичного значення будь-яких трьох чисел. Алгоритм передбачає послідовність виконання таких дій:

Задати три числа, наприклад, x, y, z. Обчислити їх середнє арифметичне за формулою $s=\frac{x+y+z}{3}$. Вивести значення результату, яке отримала змінна s.

Тепер наведемо приклад програми, що реалізує наведений алгоритм:

```
program my_prog;
 uses crt;
 var x,y,z,s:     real;
         k:       char;
 begin
   ClrScr;
   TextColor(Green);
   GoToXY(30,10);
```

```
write ('Введіть три будь-яких числа:');
readln (x,y,z);
s:=(x+y+z)/3;
GoToXY(30,15);
write ('Середнє арифметичне чисел ',x:7:3,y:7:3,z:7:3);
GoToXY(45,16);
write (s:10:5);
k:=ReadKey;
end.
```

Питання для самоконтролю:

1 *Які стандартні процедури забезпечують введення інформації у Паскалі?*
2 *Як відбувається введення інформації?*
3 *Які стандартні процедури забезпечують виведення інформації у Паскалі?*
4 *Що може бути елементами процедури виведення у Паскалі?*
5 *Для чого у Паскалі використовується форматування змінних?*
6 *Що розуміють під модулями у Паскалі?*
7 *Як скористатися функціями та процедурами того чи іншого модуля?*
8 *Які режими роботи моніторів підтримують Паскаль-програми?*
9 *Назвіть основні функції та процедури модуля* ***CRT****.*
10 *Яким чином можна організувати затримку виконання програми?*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

У цих розділах буде зроблена спроба застерегти вас від типових помилок, яких допускають майже усі початківці. Будуть вони обов'язково і у вас, але звертайтеся до цих розділів і скоріше за все ви знайдете в них відповіді майже на всі запитання.

І ще одне зауваження. Частину помилок ви будете одержувати під час компіляції програми – це частіше за все помилки синтаксичного характеру. В результаті компіляції програми буде вказано на розташування таких помилок. Коли ж усі вони будуть виправлені, можуть з'явитися помилки на етапі виконання вашої програми – це вже арифметичні або ж логічні помилки. Їх аналізувати важче, але можливо.

Якщо компілятор видав помилку

Error 85: «;» expected,

то це означає, що перед символом, на який вказує курсор, відсутній символ ";". Ця помилка, швидше за все, зроблена вами в кінці попереднього рядка.

Якщо ви ***зробите*** помилку в записові самого оператора Паскалю, то компілятор зреагує на це повідомленням

Error 113: Error in statement .

Якщо при ***компіляції*** вашої програми було видане повідомлення про помилку

Error 4: Duplicate identifier,

то це може означати, що ім'я програми та деякої змінної або двох змінних однакові.

Якщо компілятор видасть повідомлення про помилку

Error 3: Unknown identifier,

то це означає, що вказане ім'я змінної не описане в розділі змінних.

При непарній кількості апострофів в процедурі **write** компілятор видає повідомлення про помилку з таким коментарем:

Error 8: String constant exceeds line,

що означає «рядкова константа перевищує розміри рядка».

Будьте уважні при введенні дуже довгих рядків. На такі ситуації компілятор реагує повідомленням про помилку:

Error 11: Line too long.

Якщо у кінці програми немає крапки, то буде видане повідомлення про помилку

Error 10: Unexpected end of file,

що означає «несподіване закінчення файла».

Якщо при ***компіляції*** програми не розпізнається ім'я будь-якої процедури або функції з модуля **CRT**, то в першу чергу перевірте, чи підключений цей модуль.

15. Практична робота «Введення та виведення даних»[2]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм з використанням введення та виведення даних.

1) *Визначити кількість вхідних даних відповідно до умови задачі.*
2) *Визначити кількість вихідних даних відповідно до умови задачі.*

---

[2] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Введення та виведення даних»

3) *Визначити типи вхідних та вихідних даних.*
4) *Визначити тип алгоритму.*
5) *Розробити словесний опис алгоритму.*
6) *Побудувати схему алгоритму.*
7) *Визначити послідовність введення початкових даних з використанням відповідних коментарів.*
8) *Визначити послідовність виведення результуючих даних з використанням відповідних коментарів.*
9) *Записати алгоритм мовою програмування.*
10) *Набрати текст програми, використовуючи середовище програмування.*
11) *Виконати налагодження програми, виправивши синтаксичні помилки.*
12) *Виконати програму, підготувавши систему тестів.*

16. Надання значення величині

Арифметичні операції та арифметичні вирази

З математики відомо, що існує чотири основних арифметичних дії: додавання, віднімання, множення та ділення.

Цілком очевидне означення:

Арифметичними називатимемо такі вирази, які записуються за допомогою арифметичних операцій і внаслідок обчислення яких одержуються числові значення.

Поки що це означення не дало нової інформації в порівнянні з математикою та фізикою. Це настільки очевидно, що незрозуміла необхідність в цьому означенні! Не кваптеся, трохи пізніше виявиться, що у Паскалі використовуються не тільки арифметичні вирази.

Записуючи арифметичні вирази в словесному представленні алгоритму, у схемах, мовою псевдокодів, ми можемо використовувати математичні правила їх представлення.

Наприклад, для визначення швидкості руху деякого об'єкта формула виглядатиме так: $V = \frac{S}{T}$. А для обчислення дискримінанта квадратного рівняння формула буде такою: $d=b^2-4ac$.

Однак, при записові арифметичних виразів будь-якою мовою програмування треба дотримуватись певних домовленостей. Розглянемо їх.

У програмуванні так само, як і у математиці, визначений пріоритет виконання арифметичних дій, тобто визначається, яким діям надається перевага перед іншими під час обчислення значення арифметичного виразу. Наведемо арифметичні операції Паскалю саме в порядку зменшення їх пріоритетності:

***** , **/** – множення та ділення;
div – частка від ділення націло двох цілих чисел;
mod – остача від ділення націло двох цілих чисел;
+ , **–** – додавання та віднімання.

Ви звернули увагу на те, що у Паскалі відсутня операція піднесення до степеня? І це на перший погляд здається дуже незручним. З часом до такого недоліку швидко звикають і виявляється, що без цієї операції можна спокійно обійтися.

Перейдемо до ознайомлення з новими арифметичними операціями Паскаля, яких ви не зустрічали в математиці.

Операції **div** та **mod** виконуються над величинами тільки цілого типу. Результатом виконання цих дій є також цілі числа.

Принцип їх використання краще за все можна пояснити на прикладах.

10 **div** 3 = 3 – тобто в результаті одержимо відповідь на запитання, скільки разів число 3 входить у 10. Наведемо ще такі приклади: –10 **div** 3 = –3, 10 **div** –3 = –3, –10 **div** –3 = –3. Можна зробити висновок, що у разі, коли величини, над якими виконується операція **div** мають однакові знаки, то результат отримується додатній. У протилежному випадку – від'ємний.

10 **mod** 3 = 1 – тобто в результаті такої дії одержимо відповідь на запитання, скільки залишиться від числа 10, якщо вилучити з неї всі трійки. Розглянемо ще такі приклади: –10 **mod** 3 = –1, 10 **mod** –3 = 1, –10 **mod** –3 = –1. Аналіз наведених прикладів дає можливість зробити висновок, що на знак результату впливає лише знак величини, над якою виконуєтьсяя операція **mod**.

Розбираючи операції **div** та **mod**, ми вже торкнулися питання типів величин, над якими вони виконуються. Слід звернути увагу і на інші арифметичні операції.

Операції «*», «/», «+», «–» виконуються над величинами як цілого, так і дійсного типу. Питання лише в аналізі типу результату. Якщо операції «*», «+», «–» виконуються над величинами, що мають цілі значення, то і отриманий результат буде цілим числом.

Якщо ж хоча б одна з величин, над якими виконується дія, буде мати дійсний тип, то і результат буде дійсного типу. Виняток складає операція ділення «/». Якого б типу не були величини, над якими вона виконується, результат завжди матиме дійсний тип.

Ця інформація стане нам корисною надалі, коли доведеться використовувати арифметичні вирази для обчислення значень змінних.

Арифметичні вирази, як і всі інші конструкції мов програмування, вводяться з клавіатури підряд, в один рядок. Тому зразу ж зрозуміла доречність використання дужок для задання необхідної послідовності виконання дій у виразі.

Розглянемо арифметичний вираз у математичній формі та у записі на Паскалі:

$$\frac{a+b}{c+d} \Rightarrow (a+b)/(c+d)$$

А запису на Паскалі $a+b/(c+d)$ відповідає такий математичний вираз: $a+\dfrac{b}{c+d}$.

Перевірте себе ще на таких прикладах, записавши відповідний кожному з них математичний вираз:

$$a+b/c+d,\ \ (a+b)/c+d,\ \ a+(b/c)+d.$$

Набір стандартних функцій для стандартних типів

В арифметичних виразах ще можуть використовуватися стандартні функції. Деякі з них вам відомі з математики: $\sin x, \cos x, tgx$ і т.і. Правда, у Паскалі поняття функції більш широке, але про це мова буде йти пізніше. А поки що ми поговоримо про стандартні функції, тобто такі функції, які розуміє Паскаль без додаткових пояснень та описів. Пріоритетність обчислення функцій найвища. Отже, якщо в арифметичному виразі використовуються функції, то спочатку буде обчислене їх значення, а потім над цими результатами будуть виконані інші дії. Серед стандартних функцій у Паскалі є не тільки тригонометричні. І ще одна домовленість – у мовах програмування аргументи функції вказуються у дужках. Тобто в математиці ви пишете $\sin x$, а у Паскалі треба вказувати $\sin(x)$.

А тепер наведемо список стандартних функцій Паскаля із зазначенням типів їх аргументів та результатів.

sin(x) – синус;
cos(x) – косинус;
arctan(x) – арктангенс;
(аргументи тригонометричних функцій задаються в радіанах)

exp(x) – e^x (експонента);
ln(x) – $\ln x$ (логарифм натуральний);
abs(x) – $|x|$ (модуль);
sqrt(x) – $\sqrt{x}$;
sqr(x) – x^2.

Для функцій від sin(x) до ln(x) аргументами можуть бути цілі або дійсні значення, а результати завжди дійсні. Для решти функцій тип результату співпадає з типом аргумента.

trunc(x) – ціла частина x, тобто відкидається дробова частина; x – дійсне, результат – цілий. Наприклад, **trunc**(–6.7)= –6.

frac(x) – дробова частина x; x – дійсне, результат – дійсний. Наприклад, **frac**(4.5)=0.5, **frac**(–4.5)=–0.5.

round(x) – заокруглення до цілого значення; x – дійсне, результат – цілий. Наприклад, **round**(4.2)=4, **round**(4.5)=5, **round**(–4.2)= –4, **round**(–4.5)= –5.

odd(x) – визначення непарності x; x – ціле, результат – **true** або **false**. Наприклад, **odd**(2)=**false**, **odd**(5)=**true**.

int(x) – повертає цілу частину аргумента, що не перевищує аргумент (результат дійсний). Наприклад, **int**(3.7)=3, **int**(–3.7)= –3.

Pi – повертає значення числа π; аргументів немає.

Тепер час запропонувати вам математичну формулу, за допомогою якої можна піднести будь-яке число до будь-якого степеня:

$$x^y = e^{\ln x * y}.$$

Це відповідає такому паскалевському запису:

exp(ln(x)*y).

Вказівка надання значення. Оператор присвоювання

Ясна річ, якщо у програмі буде обчислене значення арифметичного виразу, то його необхідно десь запам'ятати для подальшого використання. Для цього існує оператор присвоювання.

Загальний вигляд оператора:

<ім'я змінної> := <вираз> .

Операцію присвоювання можна ще назвати операцією заміщення. Дію n:=m можна прокоментувати таким чином: значення змінної n повинно бути замінено поточним значенням змінної m .

Знак «=» слід відрізняти від знака «:=». Перший означає порівняння, умову, яку можна перевірити.

Логічно використовувати його із знаком запитання: «n=m?». Другий знак «:=» означає дію, яку потрібно виконати.

При виконанні цього оператора спочатку за заданою формулою в правій частині виконуються обчислення при поточних значеннях змінних, що входять у неї, а потім отриманим результатом замінюється попереднє значення змінної, що вказана зліва.

Опираючись на принцип виконання оператора присвоювання, зрозуміло, чому зліва може стояти лише ім'я змінної величини: адже отримане значення записується у ту частину оперативної пам'яті комп'ютера, яка виділена для цієї змінної.

При виконанні операції присвоювання важливим є співпадання типів змінної величини в лівій частині оператора і виразу, що обчислюється в правій його частині. Це пояснюється відображенням виконання оператора присвоювання на пам'ять комп'ютера: типи повинні бути однаковими. Наприклад:

p:=(a+b+c)/3;

a:=**sin**(x)+**sin**(y);

d:=**sqrt**(**sqr**(b)-4*a*c).

Але вирази, що беруть участь в операторі присвоювання, можуть бути не лише арифметичними. Наприклад, правомірні і такі присвоювання:

ch:='a';

text:='алгоритм';

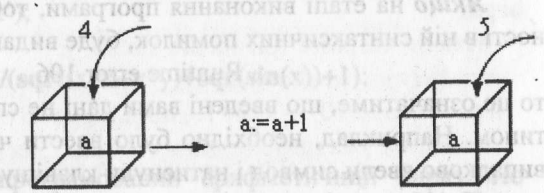
flag:=**true**.

Перший приклад присвоює змінній значення типу **char**, другий – **string**, третій – **boolean**. Пізніше ми будемо розглядати питання обчислення виразів типу **boolean**.

Операція збільшення *a* на 1 позначається таким чином: **a:=a+1**. Її слід читати так: «*a* замінити на *a*+1». Оскільки це дуже важливо розуміти, щоб не робити зайвих помилок, продемонструємо роботу саме цього оператора схематично (мал.15).

З малюнка бачимо, що спочатку в області пам'яті, яка відведена змінній a, знаходилося поточне значення 4.

Виконання оператора a:=a+1 відбувалося поетапно таким чином:



Мал. 15

- при обчисленні виразу справа було взято поточне значення змінної *a* (число 4) і до нього додалося число 1;
- одержане значення 5 було поміщене в ту область пам'яті, що відведена змінній *a*. При цьому попереднє значення змінної a втрачається.

При повторному виконанні оператора **a:=a+1** змінна **a** отримає вже значення 6.

Цікаво розглянути такі послідовності операторів присвоювання:

1) m:=n, n:=k;

2) n:=k, m:=n.

Ці послідовності операторів присвоювання різні стосовно збереження значення змінної *n*. В першому випадку значення змінної *n* було збережене у змінній *m*. В другому випадку значення змінної *n* було втрачено перш, ніж його можна було використовувати для заміщення змінної *m*. Таким чином остання послідовність операторів присвоювання фактично аналогічна такій: n:=k, m:=k.

Питання для самоконтролю:

1 *Що називають арифметичними виразами?*
2 *Назвіть арифметичні операції Паскаля в порядку їх пріоритетності.*
3 *Над якими величинами можна виконувати операції* ***div*** *та* ***mod****?*
4 *Які правила запису арифметичних виразів у Паскалі?*
5 *Як на машинному рівні виконується оператор присвоювання?*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Якщо в арифметичному виразі не вистачає дужок, то компілятор Паскаля може видати повідомлення про помилку:

Error 89: Unexpected «(»,

або ж

Error 89: Unexpected «)».

Подібне повідомлення про помилку також видається, якщо десь не вистачає символів «;» або «:=».

При неспівпаданні типу змінної і значення, якого вона набуває в результаті обчислення, буде видане повідомлення про помилку

Error 26: Type mismatch.

Якщо на етапі виконання програми, тобто при повній відсутності в ній синтаксичних помилок, буде видане повідомлення

Runtime error 106,

то це означатиме, що введені вами дані не співпадають з описаним типом. Наприклад, необхідно було ввести числове значення, а ви випадково ввели символ і натиснули клавішу «Enter».

Подібне повідомлення про помилку, але з іншим кодом, буде видане у випадку, коли ви захочете виконати ділення на вираз, що набуває значення 0, або добути корінь з від'ємного числа. У таких ситуаціях треба ретельно проаналізувати початкові дані.

17. Складання алгоритмів з використанням присвоювання

Приклад 1. Розв'яжемо задачу обчислення значення змінної z за заданими значеннями x та y, де залежність змінної z від x та y задається формулою: $\dfrac{x+\dfrac{y}{\sin x^2+10}}{\sqrt{x^2+y^2+\sin^2 x+1}}$

Словесний алгоритм розв'язання наведеної умови задачі буде таким:

Ввести початкові значення для x та y.

Обчислити значення змінної z за формулою $\dfrac{x+\dfrac{y}{\sin x^2+10}}{\sqrt{x^2+y^2+\sin^2 x+1}}$.

Вивести обчислене значення z.

Перш ніж навести текст програми, що відповідає алгоритму обчислення значення арифметичного виразу, проаналізуємо задану формулу. При виконанні лінійних програм важливо переконатися, що в результаті обчислення арифметичного виразу не виникне ситуації ділення на 0, добування кореня з від'ємного числа тощо. В даній формулі таке дійсно неможливе. Як бути у випадку, коли таке може статися, ми з вами розглянемо пізніше при знайомстві з розгалуженими алгоритмами.


```
program proba;
 var x,y,z: integer;
 begin
   writeln('Задайте x, y:');
   readln(x,y);
   z:=(x+y/(sin(sqr(x))))/(sqrt(x*x+y*y)+sqr(sin(x))+1);
   writeln('z=',z:0:3)
 end.
```

Проаналізуємо запрограмований арифметичний вираз. По-перше, кількість відкритих та закритих дужок повинна бути рівною. По-друге, ніяка закрита дужка не повинна передувати відкритій. Це попередній синтаксичний аналіз виразу. А тепер необхідно провести семантичний аналіз. Це можна зробити лише тестуванням: підібрати коректні і зручні початкові дані, для яких неважко виконати обрахунок вручну. Для нашого випадку найзручніша перевірка для x=y=0. Результат повинен бути z=0.1.

Зверніть увагу на такі фрагменти формули: **sin**(**sqr**(x)) та **sqr**(**sin**(x)). Обчислення функцій відбувається з середини, тобто у першому виразі спочатку буде обчислено значення $x2$, а потім від отриманого значення взяте **sin**, а другий – навпаки. У фрагменті **sqrt**(x*x+y*y) застосований інший варіант обчислення квадратів значень змінних. Обидва способи правомірні.

Приклад 2. Серед арифметичних операцій, що визначені в Паскалі, є дві, нетипові для математики. Це операції з цілими числами **div** та **mod**. Вони в деяких випадках бувають дуже корисними при обчисленні результатів.

Наприклад, розглянемо алгоритм визначення суми цифр тризначного числа x. Для цього треба виділити окремо його цифри за такою послідовністю дій:

Цифру наймолодшого розряду можна визначити за допомогою операції x **mod** 10.

Цифру десятків можна обчислити за два кроки: спочатку відкинути останню цифру і в отриманому двозначному числі вже відомою операцією **mod** визначити його цифру наймолодшого розряду. Послідовність операцій буде такою: (x **div** 10) **mod** 10.

Старшу цифру тризначного числа можна отримати, поділивши його націло на 100: x **div** 100.

Запишемо програму, що реалізує наведений алгоритм:

```
program proba_1;
   var x,rez: integer;
 begin
   writeln('Задайте тризначне число:');
   readln(x);
   rez:=(x mod 10)+((x div 10) mod 10)+(x div 100);
   writeln('Сума цифр заданого числа: ',rez)
 end.
```

В даній програмі ми поки що не можемо контролювати коректність введення саме тризначного числа. Це на совісті користувача програми. Але для всіх чисел, менших за тризначне, програма все ж таки буде рахувати результат коректно.

18. Практична робота «Створення лінійних програм»[3]

За наведеним сценарієм виконайте завдання по створенню та налагодженню лінійних алгоритмів та програм.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму.*
6) *Побудувати схему алгоритму.*
7) *Визначити послідовність введення початкових даних з використанням відповідних коментарів.*
8) *Визначити послідовність виведення результуючих даних з використанням відповідних коментарів та необхідність їх форматування.*
9) *Записати алгоритм мовою програмування.*

[3] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Створення лінійних програм»

10) *Набрати текст програми, використовуючи середовище програмування.*
11) *Виконати налагодження програми, виправивши синтаксичні помилки.*
12) *Виконати програму, підготувавши систему тестів.*

19. Тематична атестація «Програма. Мова програмування»

Для виконання тематичної атестації необхідно:

знати

- означення та класифікацію мов програмування;
- які мови програмування відносяться до процедурних;
- що таке структурне, логічне, об'єктно-орієнтоване, візуальне програмування, їх основні поняття та принципи;
- що таке системи програмування;
- суть трансляції програм, відмінність між інтерпретацією та компіляцією;
- особливості та призначення мови програмування Паскаль;
- алфавіт мови програмування, суть синтаксичного та семантичного аналізу програми;
- основні поняття мови програмування;
- форми представлення дійсних чисел в мовах програмування;
- структуру Паскаль-програми;
- що таке величина та її основні характеристики;
- стандартні типи даних у Паскалі;
- вказівки введення та виведення інформації та їх опис мовою програмування;
- правила запису арифметичних виразів;
- вказівку надання значення та загальний вигляд оператора присвоювання;
- принцип виконання операції надання змінній значення у відображенні на пам'ять комп'ютера;

вміти

- визначати класифікацію мов програмування;
- користуватися можливостями інтегрованого середовища програмування для створення, редагування, налагодження та виконання програм;

- користуватися процедурами та функціями модуля CRT для обробки інформації в текстовому режимі;
- користуватися стандартними функціями мови програмування, що визначені для стандартних типів;
- коректно записувати арифметичні вирази;
- розв'язувати задачі з використанням введення початкових даних, обчислення значень змінних та виведення результуючих даних;
- тестувати складені програми;
- аналізувати синтаксичні та семантичні помилки.

Алгоритми з розгалуженням та повторенням

20. Логічні вирази. Вказівка розгалуження

Крім арифметичних виразів, в алгоритмізації розглядається ще один тип виразів. Він називається логічним.

*Логічними виразами називаються такі вирази, внаслідок обчислення яких одержуються логічні значення «**true**» або «**false**» («так» або «ні»).*

Невеличке знайомство із стандартним типом змінних **Boolean**, які можуть набувати лише двох значень **False** та **True**, у нас вже відбулося. Отже, саме такий тип і набувають результати обчислення логічних виразів.

Логічні вирази поділяються на прості та складені.

Простими логічними виразами називаються такі, які записуються за допомогою знаків співвідношень «<», «>», «<=», «>=», «=» та «<>».

Приклади логічних виразів можуть здатися вам простими:

a+b > c+d,

n<>m,

x=y.

Порівняйте тепер призначення символів «:=» та «=»!

Зверніть також увагу на те, що спочатку виконуються арифметичні дії, а вже потім порівняння одержаних результатів.

*Складеними логічними виразами називаються такі, які складаються з простих виразів, об'єднаних логічними операціями «**and**», «**or**», «**not**».*

Складені логічні вирази є не менш цікавими і з точки зору природної логіки цілком зрозумілими.

Наведемо приклади.

З математики вам відомі такі записи:

$x \in [a,b]$ та $x \notin [a,b]$.

Спробуємо записати їх у вигляді логічних виразів

(x>=a) **and** (x<=b),

(x<a) **or** (x>b) або **not**((x>=a) **and** (x<=b)).

При записуванні складених логічних виразів прості логічні вирази обов'язково беруться у круглі дужки!

Цікаво, чи можна записати простий логічний вираз n<>m у вигляді складеного? Виявляється, можна: **not** (n=m)!

Визначимо правила, за якими обчислюються значення складених логічних виразів.

Для цього існують таблиці істинності (табл.5):

Таблиця 5

A	B	A **and** B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A **or** B
0	0	0
0	1	1
1	0	1
1	1	1

A	**not** A
0	1
1	0

Для зручності у цих таблицях цифра «0» означає **false**, а цифра «1» – **true**. Наведені таблиці можна перефразувати таким чином.

***Логічна операція* and *дає результат* true *тоді і тільки тоді, коли обидва операнди мають значення* true.**

***Логічна операція* or *дає результат* true *тоді, коли хоча б один операнд має значення* true.**

Логічна операція* not *завжди дає результат, протилежний значенню її операнда.

Ми вели розмову про обчислення значень логічних виразів. Зрозумілим є запитання: «А де їх можна використовувати?»

По-перше, використання логічних виразів так, як і арифметичних, можливе в операторі присвоювання.

Наприклад:

```
logical_1:=a>b;
logical_2:=(N<=x) and (x<=M);
flag:=false.
```

Умовою безпомилкового виконання таких операторів є співпадання типів, тобто змінні в лівій частині цих операторів повинні бути описані типом **boolean**.

По-друге, результат обчислення логічних виразів «**true**» та «**false**» можна ще трактувати як «так» та «ні». Це наводить на думку про використання логічних виразів для визначення оцінки деякої ситуації, що склалася, і прийняття рішення про те, що робити далі.

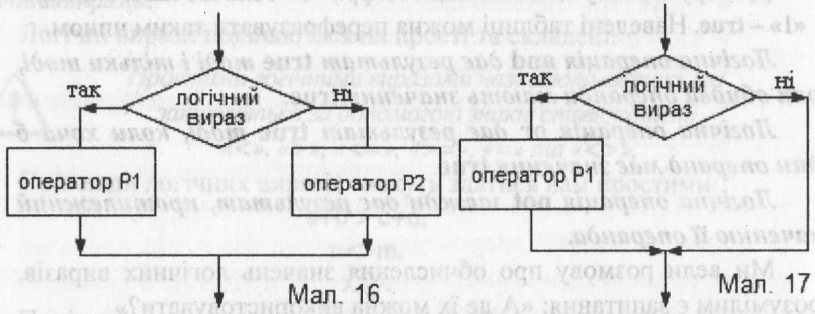
Запис алгоритмів з вказівками розгалуження

Уявіть собі, що ви за кермом автомобіля і перед вами стоїть вибір подальшого руху: їхати поганою, але коротшою дорогою, або ж гарною, але довшою. Звичайно, що вибір буде залежати від певних умов: по-перше, чи є зайвий час, по-друге, хто хазяїн автомобіля?

Схожу проблему завжди вирішують оператори розгалуження.

Загальний вигляд повного оператора умовного переходу:
***if** <логічний вираз> **then** P1 **else** P2,*
*де логічний вираз – може набувати одне з двох значень **true** або **false**, P1 та P2 – це оператори.*

Робота оператора умовного переходу не викликає ніяких труднощів. Цей оператор використовує результат обчислення логічного виразу для вибору того чи іншого шляху наступного виконання алгоритму – виконання оператора P1 або оператора P2. Після цього робота алгоритму продовжується далі за вказаними операторами.



Мал. 16

Мал. 17

Схема алгоритму повного оператора умовного переходу (мал.16) наочно демонструє, що після аналізу значення логічного виразу буде вибраний лише один з наступних напрямків виконання алгоритму (P1 або P2), після чого цей алгоритм буде виконуватися далі. Схема алгоритму скороченої форми оператора умовного переходу (мал.17) так само дуже схожа на попередню.

*Загальний вигляд скороченого оператора умовного переходу: **if** <логічний вираз> **then** P,*
де значення параметрів такі самі, як і в повній формі.

На схемі алгоритму дуже добре видно різницю між двома формами умовного оператора: в першій – повній – незалежно від значення логічного виразу якісь дії обов'язково будуть виконані, а вже потім продовжено виконання алгоритму далі, у другій – скороченій – у випадку, коли логічний вираз набуде значення **true**, будуть виконані якісь дії, а потім продовжено виконання алгоритму, а у випадку, коли логічний вираз набуде значення **false**, алгоритм зразу ж буде продовжено далі.

Розширимо поняття оператора у Паскалі. Досі ми з вами мали справу лише з простим оператором присвоювання та оператором умовного переходу. А що робити, коли після службових слів then або else нам потрібно вказати не один такий оператор, а декілька? Для такого випадку у Паскалі введене поняття складеного оператора.

*Складеним оператором називають послідовність декількох операторів, розділених символом «;» та взятих в операторні дужки **begin ... end**.*

Питання для самоконтролю:

1 *Що називають логічними виразами? На які два типи вони поділяються?*
2 *Які знаки співвідношень використовуються для запису простих логічних виразів?*
3 *Назвіть логічні операції.*
4 *Поясність використання логічних операцій за таблицями істинності.*
5 *Що називають складеним оператором? Які правила його запису?*
6 *Яким чином організоване розгалуження у Паскалі?*
7 *Чим відрізняються повна та скорочена форми оператора умовного переходу?*
8 *Запишіть загальний вигляд повної форми розгалуження.*
9 *Запишіть загальний вигляд скороченої форми розгалуження.*
10 *Намалюйте схеми алгоритмів обох варіантів розгалуження.*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Якщо ви використали в операторах розгалуження непарну кількість операторних дужок, то може бути видане повідомлення про одну з двох помилок:

Error 85: «;» expected

або

Error 113: Error in statement .

Якщо перед службовим словом **else** в операторі умовного переходу стоїть «;», то курсор буде встановлено на цьому службовому слові і буде видане повідомлення про помилку:

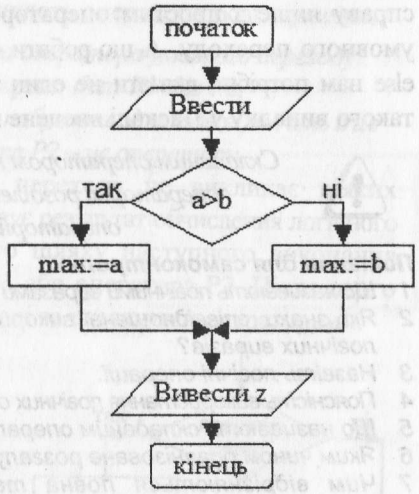
Error 113: Error in statement .

Якщо в операторах умовного переходу в окремих блоках відсутні деякі операторні дужки, то про це повідомить та сама помилка Error 113.

21. Складання алгоритмів з простими розгалуженнями

Приклад 1. Тепер вже час переходити до прикладів. Розглянемо алгоритм пошуку найбільшого з двох заданих чисел A та B.

Побудуємо алгоритм сформульованої задачі. На цей раз оберемо схематичне представлення алгоритму, як більш наочний спосіб представлення саме розгалужених типів алгоритмів (мал.18).



Мал.18

Програма, що відповідає наведеному алгоритму, може бути такою:

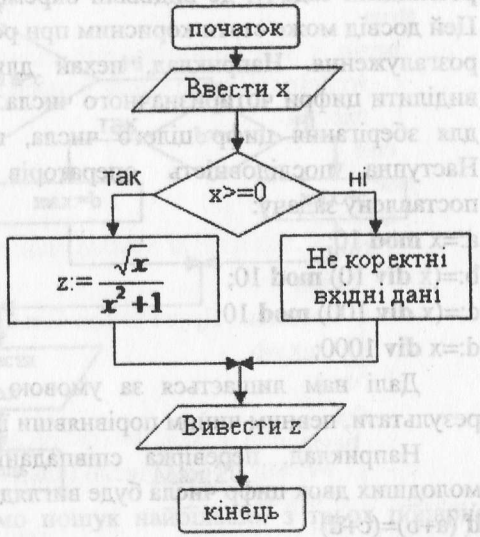
```
program max_A_B;
 var a,b,max: real;
begin
   write ('Задайте два будь-яких числа:');
   readln (a,b);
   if a>b
     then
      begin
         writeln ('Перше число більше за друге.');
         max:=a
      end
     else
      begin
         writeln ('Друге число більше або дорівнює першому.');
         max:=b
      end
   writeln ('Це число - ',max:10:5);
   readln
end.
```


Розглянемо детальніше наведену програму. По-перше, у ній чітко спостерігається принцип «вкладеності» операторів, тобто сходинкова структура. Наочність такої програми явно виграє. Для цього в ній навіть з'єднані вертикальними лініями оператори різного рівня. По-друге, перед закриваючою операторною дужкою (**end**) не стоїть символ «;». І дійсно, ви ж не ставите кому в тексті перед закриваючою дужкою, коли перераховуєте в ньому в дужках декілька слів? Хоча в Паскалі це не є помилкою. За тих самих обставин не поставлений і символ «;» після процедури **readln**.

Спробуйте відповісти на таке запитання: при виконанні якої умови буде виконаний оператор max:=b? Дійсно, за умовою, протилежною a>b, тобто при виконанні умови a<=b!

Приклад 2. Розглянемо приклад складання алгоритму обчислення значення виразу: $z = \frac{\sqrt{x}}{x^2+1}$

На перший погляд це лінійний алгоритм. Але, проаналізувавши можливі значення початкових даних, можна побачити, що не для всяких значень x вираз буде обчислено. Якщо не зробити перевірки на відсутність від'ємних значень для x, то отримати кінцевий результат не завжди можливо. Таким чином ми дійшли висновку, що даний алгоритм є розгалуженим і ознакою цього є наявність розгалуженого елемента, що перевіряє на не від'ємність значення змінної x (мал.19):



Мал.19

З точки зору програмування цей алгоритм можна охарактеризувати, як алгоритм обчислення значення виразу із захистом від введення некоректних даних. Якщо не зробити перевірку вхідних значень, то компілятор за певних некоректних умов може видати повідомлення про відповідну системну помилку. Застосувавши для цього розгалуження, можна запобігти таким неприємностям. Програма виглядатиме так:

```
program sequrity;
 var x: real;
 begin
   write ('Задайте число:');
   readln(x);
   if (x>=0)
     then writeln(sqrt(x)/(sqr(x)+1))
     else writeln('Початкові дані некоректні')
end.
```

Приклад 3. При розборі лінійних алгоритмів ми з вами розглядали задачу, де виділяли окремо цифри тризначного числа. Цей досвід може стати корисним при розв'язуванні деяких задач на розгалуження. Наприклад, нехай для певної задачі необхідно виділити цифри чотиризначного числа. Визначимо змінні *a*, *b*, *c*, *d* для зберігання цифр цілого числа, що міститься в змінній *x*. Наступна послідовність операторів присвоювання вирішить поставлену задачу:

```
a:=x mod 10;
b:=(x div 10) mod 10;
c:=(x div 100) mod 10;
d:=x div 1000;
```

Далі нам лишається за умовою задачі обробити отримані результати, певним чином порівнявши їх значення.

Наприклад, перевірка співпадання суми старших двох і молодших двох цифр числа буде виглядати так:

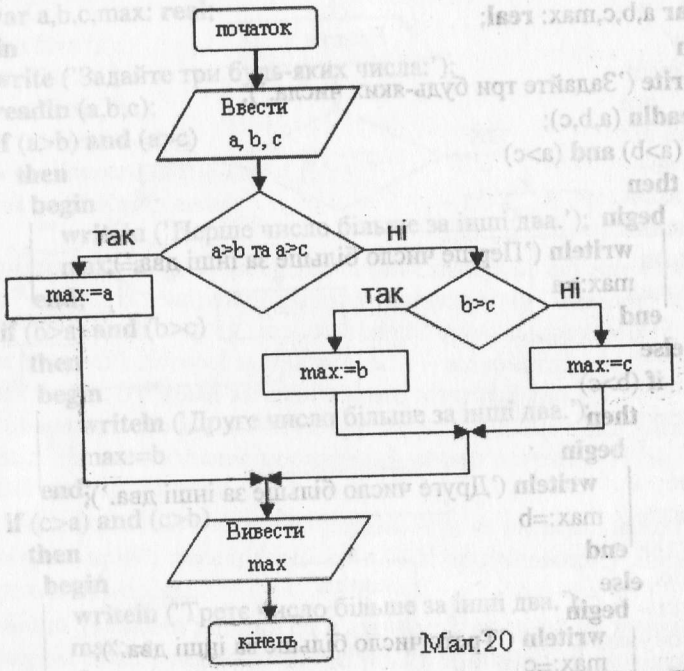
```
if (a+b)=(c+d)
   then writeln('Суми збігаються')
   else writeln('Суми не збігаються');
```


Питання для самоконтролю:

1. *Яким чином можна «захистити» програму від можливих збоїв при обчисленні арифметичних виразів?*
2. *Яким чином за допомогою арифметичних операцій для цілого ділення можна виділити окремі цифри заданого цілого числа?*
3. *Наведіть власний приклад алгоритму з повною формою розгалуження.*
4. *Наведіть власний приклад алгоритму із скороченою формою розгалуження.*
5. *Намалюйте схеми алгоритмів обох варіантів розгалуження.*

22. Складання алгоритмів з використанням вкладених розгалужень

Настав час прикладу, який продемонструє використання складених логічних виразів.



Мал. 20

Приклад 1. Розглянемо пошук найбільшої з трьох попарно різних величин a, b, c. Розберемо можливі варіанти отримання відповіді на поставлену задачу:

- найбільшим значенням є a, якщо воно більше за b та c;

• якщо твердження п.1) не справджується, то більшим значенням є b за умови, що воно більше за c;

• якщо не справджуються умови п.п.1)-2), то найбільшим буде значення c.

Мовою схем цей алгоритм буде виглядати так, як зображено на мал.20. Ми бачимо використання двох вкладених розгалужень. Внутрішнє розгалуження буде виконано тоді і тільки тоді, коли значення логічного виразу зовнішнього розгалуження матиме значення «**ні**». Це означатиме, що значення змінної *a* не є найбільшим, тому у вкладеному розгалуженні перевіряється лише значення змінної *b*. Якщо ж і її значення не є найбільшим, то лишається визнати, що найбільшим є значення змінної *c*.

У записі мовою програмування алгоритм виглядатиме так:

```
program max_A_B_C;
  var a,b,c,max: real;
begin
  write ('Задайте три будь-яких числа:');
  readln (a,b,c);
  if (a>b) and (a>c)
    then
      begin
        writeln ('Перше число більше за інші два.');
        max:=a
      end
    else
      if (b>c)
        then
          begin
            writeln ('Друге число більше за інші два.');
            max:=b
          end
        else
          begin
            writeln ('Третє число більше за інші два.');
            max:=c
          end;
  writeln ('Це число - ',max:10:5);
  readln
end.
```


Приклад 2. Однак, наведений алгоритм не є єдиним варіантом, що розв'язує задачу попереднього прикладу. Спробуємо записати інший варіант алгоритму пошуку найбільшої з трьох величин.

Будемо міркувати таким чином.

Для того, щоб значення змінної *a* було найбільшим, воно повинно бути більшим, ніж за *b* і за *c*.

Для того, щоб значення змінної *b* було найбільшим, воно повинно бути більшим і за *a* і за *c*.

Для того, щоб значення змінної *c* було найбільшим, воно повинно бути більшим і за *a* і за *b*.

Розробіть схему цього алгоритму самі та порівняйте обидва варіанти алгоритмів однієї і тієї ж задачі.

А допомогою вам в цьому буде наведений текст програми, як один із способів представлення алгоритму.

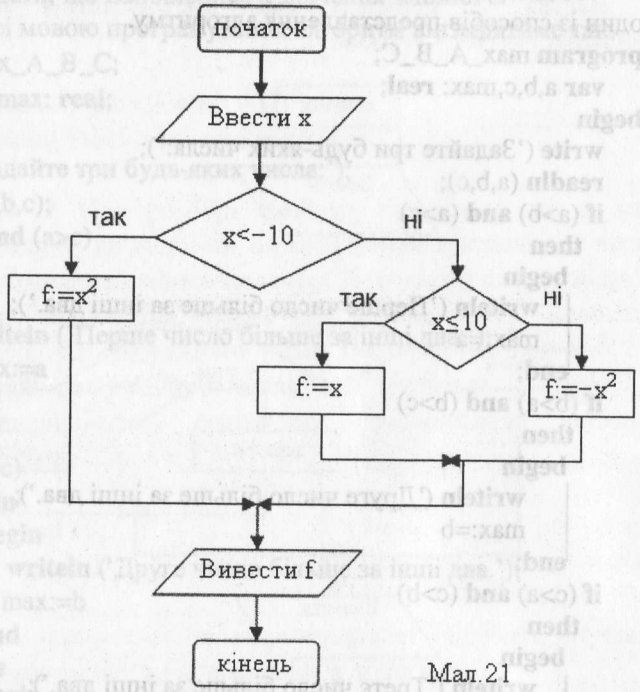
```
program max_A_B_C;
  var a,b,c,max: real;
begin
  write ('Задайте три будь-яких числа:');
  readln (a,b,c);
  if (a>b) and (a>c)
    then
      begin
        writeln ('Перше число більше за інші два.');
        max:=a
      end;
  if (b>a) and (b>c)
    then
      begin
        writeln ('Друге число більше за інші два.');
        max:=b
      end;
  if (c>a) and (c>b)
    then
      begin
        writeln ('Третє число більше за інші два.');
        max:=c
      end;
  writeln ('Це число - ',max:10:5);
  readln
end.
```

Проаналізуємо обидва алгоритми однієї і тої ж задачі пошуку найбільшого значення із трьох заданих.

Отже, у першому алгоритмі внутрішній оператор розгалуження виконається тільки тоді, коли нас «пропустить» до нього зовнішній оператор розгалуження. Цей принцип можна порівняти з ситом.

У другому варіанті алгоритму всі оператори розгалуження є послідовними. Тобто всі вони будуть виконуватися, перевіряти значення своїх логічних виразів і вирішувати, виконувати чи ні вказані в них дії. Окрім того, треба зауважити, що використані тут оператори розгалуження мають скорочену форму.

Звичайно, що з точки зору ефективності виконання алгоритму перевагу має перший варіант: у ньому може бути виконано менше перевірок для досягнення необхідного результату. Але, зрештою, все залежить від конкретної умови задачі і бачення її реалізації. Десь виграш буде на етапі виконання програми, але програш в її написанні, а десь програма буде зрозумілішою, але кількість виконуваних дій в ній буде більшою.



Мал.21

Приклад 3. Необхідно обчислити значення функції при заданому аргументі за такою формулою:

$$f(x)=\begin{cases} x^2, & \text{якщо } x<-10;\\ x, & \text{якщо } -10<=x<=10;\\ -x^2, & \text{якщо } x>10. \end{cases}$$

Алгоритм даної задачі можна записати за допомогою трьох послідовних розгалужень, що використовують неповну форму, як

це показано в попередньому прикладі. Але ми розглянемо більш ефективний з точки зору його виконання варіант алгоритму, навівши його схему (мал.21) та текст програми мовою Паскаль.

Цей алгоритм базується на використанні вкладених розгалужень, так, як це продемонстровано у прикладі 1.

```
program formula;
   var x, f: real;
begin
   writeln ('Задайте значення аргумента:');
   readln(x);
   if x<-10
      then f:=x*x
      else if x<=10
            then f:=x
            else f:=-x*x;
   writeln ('Значення функції: ',f:0:3);
   repeat until KeyPressed
end.
```

Приклад 4. Популярність діалогових програм, що дозволяють зробити вибір із запропонованого користувачу «меню», всім відома. Розглянемо задачу, яка за генератором випадкових чисел визначає виграш в лотереї таким чином: 1 – відеомагнітофон; 2 – музичний центр; 3 – комп'ютер; 4 – комп'ютерний клас.

Основою даного алгоритму є перевірка значення цілої змінної, наприклад, *ch*, яке визначає порядковий номер виграшу. Наведемо словесний вигляд алгоритму:

- Якщо значення змінної ch дорівнює 1, то виграно перший за номером приз і переходимо до п.5. У протилежному випадку переходимо до п.2.
- Якщо значення змінної ch дорівнює 2, то виграно другий за номером приз і переходимо до п.5. У протилежному випадку переходимо до п.3.
- Якщо значення змінної ch дорівнює 3, то виграно третій за номером приз і переходимо до п.5. У протилежному випадку переходимо до п.4.

- Якщо значення змінної ch дорівнює 4, то виграно четвертий за номером приз і переходимо до п.5.
- Кінець алгоритму.

Схему алгоритму цієї задачі вам надається можливість розробити самостійно. А спосіб представлення алгоритму у вигляді програми допоможе вам у цьому.

```
program prize;
  uses CRT;
  var n: byte;
      ch: char;
begin
  writeln ('Призи лотереї:');
  writeln ('1. Відеомагнітофон');
  writeln ('2. Музичний центр');
  writeln ('3. Комп'ютер');
  writeln ('4. Комп'ютерний клас');
  writeln ('Натисніть будь-яку клавішу для запуску лототрона');
  repeat until KeyPressed;
  ch:=ReadKey;
  randomize;
  n:=random(3)+1;
  write ('Ваш виграш під номером ', n,'. Це – ');
  if n=1
    then writeln ('Відеомагнітофон')
    else if n=2
           then writeln ('Музичний центр')
           else if n=3
                  then writeln ('Комп'ютер')
                  else writeln ('Комп'ютерний клас');
  repeat until KeyPressed
end.
```

Прокоментуємо деякі моменти програми. Послідовність викликів стандартних процедури і функції **randomize**; n:=**random**(3)+1 дозволяють за допомогою генератора випадкових чисел змінній n надати будь-якого випадкового значення в інтервалі від 1 до 4. Підключення модуля **CRT** дозволяє використати функції **KeyPressed** та **ReadKey**. За допомогою першої ми можемо затримати виконання програми до натискання користувачем будь-якої клавіші на клавіатурі. Але, оскільки код

цієї клавіші буде залишатися в буфері клавіатури і це стане на заваді повторного використання цієї ж функції в кінці програми, то операція ch:=**ReadKey** допоможе «вичитати» цей код із буфера і спорожнити його.

Питання для самоконтролю:

1 *У чому полягає відмінність виконання вкладених та послідовно розташованих розгалужень?*
2 *Яку роль відіграють складені логічні вирази в розгалужених алгоритмах?*
3 *Чи можна алгоритм у прикладі 3 записати за допомогою послідовних розгалужень? Якої форми вони будуть?*
4 *Чи можна алгоритм у прикладі 4 записати за допомогою послідовних розгалужень? Якої форми вони будуть?*
5 *Наведіть власний приклад алгоритму з використанням послідовних розгалужень.*
6 *Наведіть власний приклад алгоритму з використанням вкладених розгалужень.*

23. Практична робота «Програми з розгалуженнями»[4]

За наведеним сценарієм виконайте завдання по створенню та налагодженню розгалужених алгоритмів та програм.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму та побудувати його схему.*
6) *Визначити форми розгалуження, які необхідно застосувати для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання послідовних та вкладених розгалужень.*
9) *Записати алгоритм мовою програмування.*

[4] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Програми з розгалуженнями»

10) *Набрати текст програми, використовуючи середовище програмування та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*
12) *Реалізувати інші варіанти використання розгалужень для виконання сформульованої задачі та проаналізувати їх щодо кількості виконуваних дій для різних початкових даних.*

24. Вказівка повторення. Цикли

Як же ж цикли полегшують життя програмістам! Уявіть собі на хвилинку, що вам довелося б писати повторення одних і тих самих фрагментів програм багато разів! У Паскалі передбачено три різновиди операторів циклу. Всі вони різні за своїм записом і застосуванням.

Загальний вигляд оператора циклу з передумовою:

***while** <логічний вираз> **do** P,*

*де логічний вираз набуває одного з двох значень **true** або **false**, P – простий чи складений оператор.*

Цикл з передумовою працює за таким принципом.

*Повторення оператора P буде виконуватися до того часу, поки логічний вираз в операторі циклу набуватиме значення **true**. Якщо тільки на якомусь кроці циклу логічний вираз набуде значення **false**, цикл припинить свою роботу.*

Можна також зробити висновок, що цикл з передумовою може жодного разу не виконатися у випадку, коли логічний вираз зразу ж набуває значення **false.**

Одне суттєве зауваження: оскільки виконувані дії знаходяться за всіма службовими словами оператора циклу з передумовою, то не можна забувати про операторні дужки у випадку, коли тіло циклу складається з декількох таких дій.

Оператор дуже простий і зрозумілий. Схема алгоритму оператора повторення з передумовою так само не викличе ніяких непорозумінь (мал.22).

Мал. 22

Наведемо приклад програми, що передбачає захист від введення недопустимих даних користувачем:

```
program my_defence;
  var x,y,z: real;
begin
  write ('Задайте три додатних числа: ');
  readln (x,y,z);
  while (x<=0) or (y<=0) or (z<=0) do
      begin
        write ('Задайте три додатних числа:');
        readln (x,y,z);
      end;
        { продовження програми}
```

Один недолік є у цієї програми: нам довелося двічі повторити групу опитування значень змінних (**write** ... **readln**).

Спробуємо обійти і цю незручність.

Загальний вигляд оператора циклу з післяумовою:

***repeat P until** <логічний вираз>,*

де значення всіх параметрів такі самі, як і в попередньому описаному операторі.

Цикл з післяумовою працює за таким принципом.

*Повторення оператора P відбувається до того часу, поки логічний вираз отримує значення **false**. Якщо тільки на певному кроці циклу логічний вираз набуде значення true, цикл припинить свою роботу.*

Відчутна різниця між операторами **while**...**do** та **repeat**...**until**? Перший спочатку перевіряє значення логічного виразу, а потім вирішує, виконувати йому оператор, чи ні, а другий навпаки спочатку виконує вказаний оператор, а потім перевіряє, чи треба його виконувати ще раз. Ось як цю різницю наочно демонструє схема алгоритму, наведена на мал.23. Зрозуміло, що цикл з післяумовою виконуватиметься у будь-якому випадку хоча б один раз, навіть якщо логічний вираз зразу ж набуває значення **true**.

Мал. 23

А тепер надайте належне найелегантнішому захисту програми від введення недопустимих даних за допомогою оператора циклу з післяумовою:

```
program my_defence;
 var x,y,z: real;
begin
  repeat
    writeln ('Задайте три додатні. числа: ');
    read (x,y,z);
  until (x>0) and (y>0) and (z>0);
        { продовження програми}
```

Згадайте один з варіантів затримки виконання програми, що був наведений в попередніх розділах

repeat until KeyPressed .

Тепер його можна зрозуміти без додаткових пояснень.

Аби ви не подумали, що за допомогою циклів можна записувати лише повторення «стратегічного» характеру і використовувати їх для зручності роботи з програмою, наведемо приклад використання циклу для обчислення суми чисел, значення яких вводяться користувачем. Кількість чисел, що сумується, наперед невідома. Домовимося лише про пароль, який буде означати завершення введення даних. Нехай це буде число «0».

```
program suma;
 var a, sum: real;
begin
   sum:=0;
   write ('Задайте число (ознака завершення введення – число 0)');
   readln (a);
   while (a<>0) do
    begin
      sum:=sum+a;
      write ('Задайте число (ознака завершення введення – число 0)');
      readln (a);
    end;
   writeln ('Сума заданих чисел: ', sum:10:5)
end.
```


Проаналізуємо цю програму. По-перше, ми обійшлися всього двома змінними, хоча даних під час виконання програми може бути введено багато. Цей підхід називається покроковим введенням початкових даних. По-друге, перш ніж починати виконувати цикл, змінній sum було присвоєне значення «0», і лише потім на кожному кроці циклу додавалося до її попереднього значення нове, введене з клавіатури. Це дуже схоже на скриньку, з якої ми спочатку все викинули, перш ніж починати туди щось складати. Весь цей процес називається «накопиченням суми».

Виконайте самостійно таке завдання: скориставшись тим, що додавання до суми числа «0» не змінить її значення, перепишіть останню програму за допомогою оператора циклу з післяумовою.

Загальний вигляд оператора циклу з лічильником:

for <лічильник циклу>:= $X_{поч}$ to (downto) $X_{кінц}$ do P,

де лічильник циклу – змінна зчисленого типу,

Хпоч – початкове значення лічильника циклу,

Хкінц – кінцеве значення лічильника циклу,

P – простий чи складений оператор.

Службове слово **to** означає, що зміна значення лічильника циклу йде від $X_{поч}$ до $X_{кінц}$ в порядку збільшення, а **downto** – в порядку зменшення.

Цикл припинить своє виконання тоді, коли значення лічильника циклу вийде за межі вказаного відрізку $X_{поч}$ ... $X_{кінц}$.

Зверніть увагу на те, що нам невідомий крок, з яким відбувається зміна значення лічильника циклу. А це тому, що у циклі **for...to (downto)...do** на кожному наступному кроці в якості значення лічильника циклу береться наступне (або попереднє) значення. В цьому і зміст того, що лічильник циклу повинен бути зчисленого типу, тобто значення якого можна перечислити. До таких типів відносяться **integer**, **byte**, **word**, **shortint**, **longint**, **char**, **boolean**. Визначити, чи відноситься вказаний тип до зчисленого можна за таким принципом: якщо, знаючи будь-яке значення змінної цього типу, можна назвати попереднє або наступне для нього значення, то цей тип є зчисленим. Саме тому змінні типів **real** та **string** не можуть використовуватись в якості лічильників циклу **for…to (downto)…do**.

Якщо вважати кількість чисел, що сумуються, наперед відомою, то їх сума визначатиметься за допомогою такого циклу:

```
sum:=0;
write ('Задайте кількість чисел');
readln (n);
for i=1 to n do
 begin
   write ('Задайте число');
   readln (a);
   sum:=sum+a;
  end;
writeln ('Сума заданих чисел: ', sum:10:5);
```

Схема алгоритму оператора циклу з лічильником за принципом свого виконання не відрізняється від такої ж схеми алгоритму для оператора циклу з передумовою. Тому наводити її ще раз не має ніякого сенсу.

Питання для самоконтролю:

1. *Запишіть загальний вигляд оператора циклу з передумовою.*
2. *Наведіть власний приклад циклічного алгоритму з передумовою.*
3. *Який загальний вигляд має оператор циклу з післяумовою?*
4. *Наведіть власний приклад циклічного алгоритму з післяумовою.*
5. *Запишіть загальний вигляд оператора циклу з лічильником та поясніть призначення його елементів.*
6. *Наведіть власний приклад циклічного алгоритму з лічильником.*
7. *Яка характерна особливість усіх трьох операторів циклу?*
8. *Чим різняться схеми алгоритмів операторів циклу з передумовою та післяумовою?*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Використання в циклах непарної кількості операторних дужок призводить до однієї з двох помилок:

Error 85: «;» expected

або

Error 113: Error in statement .

Якщо тіло циклу типу **for...to(downto)...do** або **while...do** не взято в операторні дужки, то ви це відчуєте за одержаними результатами, тобто вдумливо підібрані тестові початкові дані не дадуть очікуваних результатів.

Якщо в циклі **for...to(dowto)...do** лічильник циклу не є зчисленим типом, то буде видане повідомлення про помилку:

Error 29: Ordinal type expected .

Якщо програма не завершує роботу, виводить одні й ті самі значення, то це може означати, що в операторах типу **while...do** або **repeat...until** не передбачили зміну лічильника циклу.

Якщо забули описати лічильник циклу в розділі змінних, то він буде невідомий програмі і одержите повідомлення про помилку:

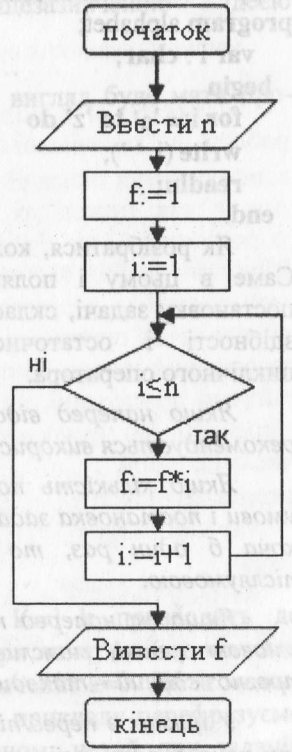
Error 3: Unknown identifier.

25. Складання алгоритмів з використанням простих повторень

Приклад 1. В математиці існує поняття факторіалу, значення якого визначається за такою формулою

$$n! = 1*2*3*4*5*...*n .$$

Тобто для отримання результату необхідно визначити добуток всіх натуральних чисел від 1 до *n*. Алгоритм цієї задачі є типовим алгоритмом накопичення суми.



Мал.24

Приклад схожого алгоритму, але накопичення суми, наведений у попередньому розділі. Треба лише зауважити, що перед початком накопичення результату змінній, яка отримуватиме значення добутку, необхідно надати значення 1. Саме таке початкове значення не змінить наступного результату. Схема алгоритму наведена на мал.24.

```
program factorial;
 var i,n,f: longint;
begin
  f:=1;
  write ('Задайте значення числа n:');
  readln (n);
  for i:=1 to n do
      f:=f*i;
  write('Значення факторіалу числа ');
  write (n:5,'дорівнює:',f:10)
end.
```

Ще раз зауважимо, що накопичення добутку відрізняється від накопи-

чення суми тим, що початкове значення обов'язково повинно бути рівним «1»!

Приклад 2. Якщо ми, на перший погляд, лише незначним чином змінимо нашу програму, ввівши виведення результату у тіло циклу, то отримаємо зовсім інший ефект. Результатом її виконання буде виведення значень факторіалу на кожному кроці. Такий принцип називається покроковим виведенням результатів.

```
for i:=1 to n do
  begin
    f:=f*i;
    writeln ('Значення факторіалу числа ', n:5,'дорівнює:',f:10)
  end;
```

Приклад 3. Для того, щоб вивести на екран монітора всі літери латинської абетки, введіть таку програму:

```
program alphabet;
  var i : char;
begin
  for i:= 'a' to 'z' do
  write (i, ' ');
  readln;
end.
```

Як розібратися, коли який оператор циклу використовувати? Саме в цьому і полягає майстерність програміста – оцінити постановку задачі, скласти алгоритм, залучити усі свої аналітичні здібності і остаточно вибрати найоптимальніший варіант циклічного оператора.

Якщо наперед відома кількість повторень тіла циклу, то рекомендується використати цикл з лічильником.

Якщо кількість повторень залежить від виконання деякої умови і постановка задачі передбачає обов'язкове виконання циклу хоча б один раз, то в цьому випадку підійде лише цикл з післяумовою.

Якщо ж наперед невідома кількість повторень циклу, і, за умовою задачі, можливе невиконання циклу жодного разу, то прогноз єдиний – підходить лише цикл з передумовою!

У циклах з перед/післяумовою створені умови повинні містити хоча б одну змінну величину, значення якої змінюється в тілі цього циклу. Інакше ви маєте нагоду створити «вічний двигун».

Приклад 4. Подекуди при складанні циклічних алгоритмів зручно використовувати так звані «перемикачі». Розглянемо два типових «перемикачі» і приклади їх використання.

Подивимось, як буде змінюватись значення змінної *k* під час виконання в циклі такого оператора k:=–k. Зрозуміло, що на кожному наступному кроці по модулю значення змінної *k* змінюватись не буде, а лише буде змінюватись на протилежний знак цієї величини.

Наприклад, при початковому значенні k=1 буде отримуватись послідовність **–1, 1, –1, 1, …**

При початковому значенні k=–1 ця послідовність буде мати вигляд **1, –1, 1, –1, …**

Застосування такого «перемикача» можна продемонструвати на прикладі.

Нехай при заданій кількості доданків *n* необхідно знайти суму 1–2+3–4+5–6+… Скориставшись вищезазначеною ідеєю представимо шукану суму у вигляді
(+1)*1+(–1)*2+(+1)*3+(–1)*4+(+1)*5+(–1)*6+…

Тепер, здається, вже зрозуміло, який вигляд буде мати програма обчислення значення шуканої суми.

```
program sum1;
  var i, s, n, k: integer;
begin
  s:=0;
  k:=1;
  write ('Задайте значення числа n:');
  readln (n);
  for i:=1 to n do
    begin
      s:=s+k*i;
      k:=-k;
    end;
  writeln ('Значення суми дорівнює:',s:10)
end.
```

Приклад 5. Виконання оператора k:=1–k призводить до отримання «перемикача» типу 0,1,0,1,… при початковому значенні k=1 і типу 1,0,1,0,… при початковому значенні k=0. Де ж можна застосувати такий «перемикач»? В якості прикладу перефразуємо задачу з попереднього пункту таким чином: нехай при заданій кількості доданків *n* необхідно знайти суму 1+3+5+7…

Перепишемо шукану суму у більш зручному вигляді

1*1+0*2+1*3+0*4+1*5+0*6+1*7+…

Програма, що буде обраховувати цю суму, матиме вигляд.

```
program sum2;
  var i, s, n, k: integer;
begin
  s:=0;
  k:=1;
  write ('Задайте значення числа n:');
  readln (n);
  for i:=1 to n do
    begin
      s:=s+k*i;
      k:=1-k;
    end;
  writeln ('Значення суми дорівнює:',s:10)
end.
```

Приклад 6. В попередніх розділах ми розглядали алгоритми роботи з цілочисловою арифметикою, тобто за допомогою арифметичних операцій **div** та **mod** виділяли окремі цифри числа. Але в тих прикладах нам була відома розрядність чисел, що задавалися. А як бути у випадку, коли розрядність невідома? Відділяти цифри від числа можна по одній, починаючи з цифри наймолодшого розряду. Залишається тільки вияснити, коли цей процес треба припинити, тобто необхідно визначити, що є ознакою завершення циклу. Зрозуміло, що відкидаючи по одній цифрі, ми дійдемо до ситуації, поточне значення числа перетвориться на 0. Перейдемо до програми, в якій ми виводитемо всі цифри заданого числа по одній.

```
program number;
  var i, n: integer;
begin
  writeln('Задайте значення числа n:');
  read (n);
  if n=0
    then writeln(0)
    else
      while n>0 do
        begin
          writeln (n mod 10);
          n:=n div 10;
        end;
    repeat until KeyPressed
end.
```


В цій програмі застосований цикл з передумовою, оскільки нам наперед невідома розрядність числа. І ще одна особливість даного алгоритму: в процесі його роботи ми втрачаємо початкове значення заданого числа. Тому, якщо нам необхідно його зберегти, треба перед початком циклу переприсвоїти початкове значення іншій змінній.

Ще одне можливе коректування в наведеній програмі: якщо необхідно, наприклад, визначити суму цифр числа і його розрядність, то для цього слід виділити відповідні змінні:

```
s:=0;
count:=0;
while n>0 do
  begin
    s:=s+(n mod 10);
    count:=count+1;
    n:=n div 10;
  end;
```

А якщо треба визначити цифру найстаршого розряду заданого числа, то доведеться так скоректувати алгоритм:

```
while n>0 do
  begin
    a:=n mod 10;
    n:=n div 10;
  end;
writeln ('Цифра найстаршого розряду числа –',a);
```

Питання для самоконтролю:

1 *Якого початкового значення повинна мати змінна, в якій надалі здійснюється накопичення суми?*
2 *Якого початкового значення повинна мати змінна, в якій надалі здійснюється накопичення добутку?*
3 *Якого значення покроково буде набувати змінна k в результаті виконання в циклі оператора k:=1–k?*
4 *Якого значення покроково буде набувати змінна k в результаті виконання в циклі оператора k:=–k?*

5 *Як за допомогою арифметичних операцій цілочислового ділення визначити всі цифри числа, розрядність якого невідома? Запишіть фрагмент програми.*

6 *Як визначити розрядність заданого цілого числа? Запишіть фрагмент програми.*

26. Складання алгоритмів з використанням вкладених повторень

Приклад 1. Складемо алгоритм задачі, що має суто математичну умову: обчислити значення виразу $\sum_{i=1}^{20}\sum_{j=1}^{30}\frac{i+j}{i*j}$.

Цей вираз читається таким чином: для всіх значень *i*, що міняються від 1 до 20, обчислити суму доданків, які в свою чергу є сумою всіх значень дробів, в яких *j* міняється від 1 до 30. Тобто для кожного значення *i* треба міняти всі значення *j* в заданому інтервалі і отримані дроби при цих значеннях *i* та *j* складати. Розписавши цей запис окремо за дробами, матимемо таку суму:

$$\frac{1+1}{1*1}+\frac{1+2}{1*2}+\frac{1+3}{1*3}+\ldots+\frac{1+30}{1*30}+\frac{2+1}{2*1}+\frac{2+2}{2*2}+\ldots+\frac{2+30}{2*30}+\ldots+\frac{20+30}{20*30}$$

Ця, не зовсім зручна на перший погляд сума дробів, представляється у вигляді дуже акуратного алгоритму, записаного у вигляді програми:

```
program suma;
 var i,j: integer;
   sum: real;
 begin
   sum:=0;
   for i:=1 to 20 do
     for j:=1 to 30 do
         sum:=sum+(i+j)/(i*j);
   writeln ('Сума:',sum:0:3);
   repeat until KeyPressed
 end.
```

Як бачимо, маємо приклад алгоритму, в якому навіть не треба вводити початкові дані.

Приклад 2. Напевно, вам відомий такий пристрій, як спідометр, і ви пам'ятаєте, як він працює. Частіше за всі інші міняється його остання цифра. Коли вона добіжить значення

«9», цифра перед нею збільшиться на одиницю, а сама вона почне мінятися знову від «0» до «9». Так само будуть мінятися і всі останні цифри спідометра.

Отже, ми маємо справу зі зміною зразу декількох значень в межах від «0» до «9». Причому зміна значень на одиницю старших цифр залежить від того, чи пробігла весь діапазон значень молодша цифра. Виявляється, що алгоритм, який буде імітувати роботу спідометра, виглядає досить просто, якщо записати його у вигляді програми:

```
program spidometer;
   uses crt;
   var i, j, k: integer;
begin
   GoToXY (45,10);
   write ('Спідометр:')
   for i:=0 to 9 do
     begin
       GoToXY (45,12);
       write (i);
       for j:=0 to 9 do
         begin
           GoToXY (48,12);
           write (j)
           for k:=0 to 9 do
             begin
               GoToXY (51,12);
               write (k)
             end;
         end;
     end;
   repeat until KeyPressed
end.
```

Вкладені цикли організуються за принципом «матрьошки»: спочатку повністю відпрацьовує самий внутрішній цикл, після цього значення лічильника зовнішнього циклу міняється на наступне, а внутрішній цикл починає свою роботу спочатку.

Питання для самоконтролю:

1 *Поясніть роботу вкладених циклів.*
2 *За яким принципом організовуються вкладені цикли?*
3 *Чи можна два вкладених цикли замінити двома послідовними циклами? Що при цьому зміниться?*
4 *Чи можна в прикладі 1 поміняти місцями зовнішній та вкладений цикли? Що при цьому зміниться?*
5 *Чи може бути, на ваш погляд, більше, ніж два вкладених цикли?*

27. Практична робота «Програми з повтореннями»[5]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм з повтореннями.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму або його фрагментів, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму та побудувати його схему.*
6) *Визначити тип вказівок повторення, необхідних для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання послідовних та вкладених вказівок повторення, наявність розгалужень.*
9) *Записати алгоритм мовою програмування.*
10) *Набрати текст програми, використовуючи середовище програмування та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*

[5] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Програми з повтореннями»

12) *Реалізувати інші варіанти використання повторень для виконання сформульованої задачі та проаналізувати їх щодо кількості виконуваних дій для різних початкових даних.*

28. Тематична атестація «Вказівки повторення та розгалуження»

Для виконання тематичної атестації необхідно:

знати

- означення логічних виразів;
- означення простих та складених логічних виразів;
- таблиці істинності для обчислення значень логічних виразів;
- суть вказівки розгалуження;
- загальний вигляд повної та скороченої форми вказівки розгалуження;
- відмінність послідовних та вкладених розгалужень;
- суть вказівки повторення;
- види операторів циклу, реалізованих в мові програмування;
- загальний вигляд оператора циклу з передумовою, післяумовою, з лічильником;
- особливості застосування різних типів операторів циклу;
- особливості використання простих та вкладених циклів;
- відмінність між послідовними та вкладеними циклами;

вміти

- наводити приклади логічних виразів та обчислювати їх значення;
- записувати загальний вигляд повної та скороченої форми вказівки розгалуження мовою програмування;
- застосовувати повну та скорочену форму вказівки розгалуження;
- коректно застосовувати послідовні та вкладені розгалуження;
- розв'язувати задачі на складання розгалужених алгоритмів;
- коректно застосовувати різні типи операторів циклу;
- коректно застосовувати послідовні та вкладені циклічні конструкції;
- будувати схеми алгоритмів з використанням розгалужень та повторень;

- розв'язувати задачі на складання циклічних алгоритмів;
- аналізувати алгоритми та їх реалізацію у вигляді програми з використанням розгалужень та повторень.

Табличні величини

29. Опис табличних величин мовою програмування

Досі ми навіть при обчисленні деякої послідовності значень обходилися декількома змінними. Але це не завжди може врятувати.

Поставимо перед собою таке завдання: нехай ми працюємо на метеостанції і в наші функції входить підрахунок щогодинного відхилення від середньодобової температури. Ясна річ, спочатку необхідно щогодини записувати значення температури, потім знайти середнє арифметичне цих значень, і тільки після цього можна обчислити відхилення кожного записаного значення від середньоарифметичного.

Ми дійшли висновку, що доведеться мати справу з 24-ма змінними величинами, в значеннях яких немає ніякої закономірності. Якими ідентифікаторами їх позначити, яким чином сумувати їх значення? Використовувати для цього набутий нами скромний досвід недоречно. Пропонується такий вихід з положення: в математиці елементи послідовності позначають таким чином:

$$a_1, a_2, ..., a_n .$$

Скористаємося схожим позначенням і ми

T[1], T[2], ... , T[24] ,

і називатимемо цю послідовність лінійною таблицею або в термінах мови програмування масивом.

Масивом називається скінчена послідовність змінних одного типу, імена яких складаються з імені масиву та покажчика, що визначає положення змінної у масиві.

Покажчик, що визначає місцеположення елемента в масиві, називається індексом.

Частіше за все в якості індексів використовують порядкові номери елементів масиву, але загалом в алгоритмізації це поняття є більш абстрактним, оскільки індекси можуть набувати і від'ємних цілих чисел.

Отже, введено новий тип – масив. Усі типи, які досі були вам відомі, називаються, ***простими***. Масив є прикладом ***структурованого типу***, тобто він в свою чергу складається з елементів іншого типу.

Можемо зробити висновок: в поставленій задачі ми маємо справу з масивом значень температур T[i], де i=1,2,3,...,24.

Як звернутися до елементів цього масиву? Для цього необхідно вказати індекс. Наприклад,

T[2], T[5], T[i], T[i+j].

Але в третьому та четвертому прикладах для визначення необхідного елемента масиву треба знати значення величин *i* та *j*. Таке загальне визначення індексу масиву є дуже потужним засобом програмування, але разом з тим і провокує можливі помилки: отриманий результат обчислення індексу масиву може виходити за межі проміжку, виділеного для індексів даного масиву.

І ще один важливий момент, який в жодному разі не можна обійти. Масиви відносяться до структур з так званим ***прямим або довільним доступом***: для того, щоб визначити окремий елемент масиву, достатньо вказати його індекс.

Тепер стало зрозумілим, як в циклі перебирати значення різних елементів масиву: для цього достатньо міняти лише їх індекси. А закон зміни індексів дуже простий: кожне наступне значення більше попереднього на одиницю. Дуже зручна закономірність!

Масив називається одновимірним, якщо для задання місцезнаходження елемента в масиві необхідно вказати значення лише одного індексу.

Це означає, що наш масив є одновимірним.

А що станеться, якщо ми сформульовану на початку задачу поширимо з однієї доби на весь місяць? Логічною буде така заміна позначення елементів:

T[1,1], T[1,2], T[1,3], ... , T[1,24],
T[2,1], T[2,2], T[2,3], ... , T[2,24],
.....
T[30,1], T[30,2], T[30,3], ..., T[30,24].

Перший індекс кожного елемента такого масиву означатиме добу, а другий – годину в цій добі. Тобто в загальному вигляді всі змінні можемо записати таким чином:

T[i,j], де i=1,2,3,...,30, j=1,2,3,...,24.

Останній приклад даних більше схожий на таблицю і носить назву прямокутної табличної величини.

Дамо йому означення мовою програмування.

Масив називається двовимірним, якщо для задання місцезнаходження елемента в масиві необхідно вказати значення двох індексів.

Запам'ятайте, що у двовимірних масивах перший індекс завжди вказує на номер рядка, а другий – на номер стовпчика елемента в цьому рядку!

Розмірність масивів у Паскалі необмежена, питання впирається лише в об'єм пам'яті, який використовується для розміщення значень масивів.

Резонним буде запитання: а як же розташовуються масиви в пам'яті комп'ютера? Пояснення для одновимірних масивів дуже просте: всі вони розташовані в пам'яті підряд. Двовимірні масиви розташовуються трохи хитріше: спочатку елементи першого рядка, потім другого і т. д.

Розташування масивів більшої розмірності пояснюється аналогічно. Наприклад, A[i,j,k].

Лишилося тільки розібратися, як пояснити програмі, що вона буде працювати з елементами, які утворюють масив значень.

Загальний вигляд опису масивів:
<ім'я змінної>: array [<межі зміни індексів>] of <тип>.

Наприклад,

```
var
A: array [1..10] of real;
B: array [1..100,1..100] of byte;
```

Зауваження. По-перше, межі індексів завжди вказуються через два символи «.». По-друге, при розподілі пам'яті в описовій частині програми під масив буде зарезервовано стільки місця, скільки передбачає вказана кількість елементів масиву вказаного типу. Тому на кроці виконання програми можна використовувати кількість елементів не більшу, ніж описана в розділі змінних. По-третє, межі зміни індексів повинні бути сталими величинами, а не змінними, інакше невідомо буде, скільки місця необхідно відвести в пам'яті під такий масив.

Зрозуміло, що при виконанні на комп'ютері програм, в яких використовуються масиви, доводиться під час виправлення можливих помилок щоразу знову і знову задавати початкові дані. А в разі використання великих масивів це введення даних буде досить громіздким. Можливості Паскаля дозволяють уникнути такої незручності.

Повернемося до початку знайомства з Паскалем, де вводилося поняття про розділ констант (**const**). В цьому розділі ідентифікаторами позначаються сталі величини. Надалі в програмі замість цих сталих величин ми будемо вказувати лише відповідні їм ідентифікатори. Зручність полягає в тому, що якщо знадобиться поміняти значення цієї константи, то достатньо це зробити тільки в розділі **const**, не чіпаючи всієї програми.

Наприклад, можна розміри масиву задати таким чином:

```
const SizeLine=100;
      SizeColumn=100;
var A: array[1..SizeLine,1..SizeColumn] of real;
.....
repeat
  read(n,m)
until (n>0) and (n<=SizeLine) and (m>0) and (n<=SizeColumn);
.....
```

Зауважте, що в розділі констант стоїть символ «=», а не «:=».

Використовуючи в програмі константи, необхідно враховувати таку особливість: їх значення не можна змінювати під час виконання програми. Тобто ідентифікаторам, що визначають константи, не можна присвоювати ніякі нові значення. Їх можна використовувати лише для обчислення інших значень.

Але ж ми вели мову про те, як незручно під час редагування програми щоразу знову задавати значення елементів масиву. Спробуємо і це зробити за допомогою розділу констант:

```
const
  A: array[1..3,1..4] of real = ((1.5, 1.2, 2.1, –4.42),
                                 (2.4, 5.7, –1, 45.4),
                                 (–1.1, 7, 45, –10) );
```

З останнього прикладу ми бачимо, що в розділі констант можна задавати ще й тип. Такі константи називаються ***типізованими***. Значення елементам масивів задаються таким чином: в одновимірних масивах вони записуються в дужках через кому, у

двовимірних значення елементів кожного рядка додатково беруться у круглі дужки. Відмінність типізованих констант від простих, для яких тип не вказується, полягає в тому, що їх значення можна міняти під час виконання програми.

Якщо вас взагалі не цікавлять конкретні дані, які будуть надані елементам масиву під час виконання програми, то скористайтеся можливостями стандартної процедури Паскаля **Randomize** та функції **Random**(n), що генерують випадкові числа. Параметр n (типу **Word**) в процедурі **Random** визначає праву межу проміжку, в якому будуть визначатися випадкові числа (ліва межа завжди 0). Функція **Random** може задаватися і без параметра. В цьому випадку вона буде генерувати дійсне число в діапазоні [0;1). А для того, щоб випадкові числа в програмі з кожним її наступним запуском не повторювалися (хоча вони і випадкові, але послідовність цих чисел постійна), скористайтеся процедурою **Randomize**, яка встановить початок відрахунку випадкових чисел в залежності від поточного стану системного годинника комп'ютера.

Фрагмент програми, що використовує випадкові числа, може виглядати таким чином:

```
randomize;
for i:=1 to n do
  a[i]:=random(100);
```

Тепер робота з масивами не буде викликати у вас незручностей.

Питання для самоконтролю:

1 *Що називається масивом?*
2 *Дайте визначення одновимірних масивів.*
3 *Дайте визначення двовимірних масивів.*
4 *Яким чином елементи масивів розташовуються в пам'яті комп'ютера?*
5 *Що називається індексом масиву?*
6 *Які типи змінних називаються простими, а які – структурованими?*
7 *Як використовується розділ констант для задання початкових даних елементам масиву?*
8 *Як використовується генератор випадкових чисел для задання початкових значень елементам масиву?*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Якщо кількість елементів описаного масиву завелика і не може бути розміщена у пам'яті комп'ютера, то ви одержите повідомлення про помилку

Error 29: Ordinal type expected .

Якщо при визначенні розмірів масиву в розділі **var** були використані ідентифікатори, які не є константами, то ви одержите повідомлення про помилку

Error 3: Unknown identifier .

Якщо індекси масиву не описані в розділі змінних, то буде видано повідомлення про помилку

Error 3: Unknown identifier .

Якщо індекс масиву виходить за межі заданих розмірів, то ви одержите повідомлення про помилку

Error 76: Constant out of range .

Якщо розмірність вказаного масиву не співпадає з його описом в розділі змінних, то ви одержите повідомлення про попередню помилку.

Якщо в результаті використання вкладених циклів ви одержуєте результат, якого не чекали, розберіться з вкладеними циклами за допомогою покрокового виконання програми (**F8**, **Ctrl+F4**).

30. Алгоритми і програми роботи з таблицями

Перше, на що треба звернути увагу при складанні алгоритмів з табличними величинами, – це введення та виведення даних. При роботі з простими змінними нам необхідно в процедурах введення-виведення перерахувати імена змінних, значення яких нас цікавлять.

При використанні масивів нам не обійтись без застосування циклів. Причому, для роботи з одновимірними масивами (лінійними таблицями) нам знадобиться організація одного циклу, а от двовимірні (прямокутні таблиці), як правило, вимагають застосування двох вкладених циклів.

Розглянемо спочатку алгоритми роботи з одновимірними масивами. Введення початкових даних виглядатиме так:

```
var i,n:   integer;
  A: array[1..100] of integer;
.....
  write ('Задайте кількість елементів масиву:');
  readln (n);
  riteln ('Введіть елементи масиву:');
  for i:=1 to n do
   begin
```

```
    write ('A[',i,']=');
    readln (A[i])
  end;
```

Зверніть увагу на те, як використані процедури введення-виведення щодо переходу на новий рядок для наступної обробки інформації. При введенні кількості елементів масиву курсор лишатиметься в тому ж рядку, де й запит на введення інформації, а після введення числа перейде на новий рядок. При введенні значень елементів масиву на екран буде виводитися коментар у вигляді імені масиву та його порядкового номера. Це дасть змогу контролювати процес введення інформації. Знову ж таки комбінація **write–readln** дасть змогу вводити числа зразу ж після коментаря, і лише потім курсор переміститься на новий рядок, але всі значення елементів масиву вводитимуться в стовпчик.

Виведення елементів одновимірного масиву майже не відрізняється від введення:

```
writeln ('Елементи результуючого масиву:');
for i:=1 to n do
  write ('A[',i,']=', A[i],' ');
writeln;
```

В наведеному прикладі виведення елементів одновимірного масиву буде відбуватись в один рядок. Якщо в першому рядку місця не вистачить, то інформація перейде на наступний, поки не завершиться виведення. Для відділення значення одного елемента від іншого використано пробіли ' '. Можна замінити пробіли форматуванням виведення.

Наприклад, для дійсних чисел: **write** ('A[',i,']=', A[i]:7:2). Але в такому випадку може статися, що зарезервованих позицій на значення числа буде замало, і вони все одно зіллються. Тому використання пробілів лишається корисним.

Можна запропонувати ще один варіант виведення одновимірного масиву, при якому контролюється кількість елементів в рядку. Нехай нам треба, щоб в рядку було розташовано 8 чисел.

Виведення виглядатиме так:

```
writeln ('Елементи результуючого масиву:');
for i:=1 to n do
  begin
    write (A[i]:0:3);
    if i mod 8 = 0 then writeln;
  end;
```


Оператор розгалуження **if** i **mod** 8 = 0 **then writeln** контролюватиме досягнення змінною **i** значення, кратного 8, і при виконанні цієї умови переводитиме курсор на новий рядок.

Отже, зв'язок виконуваної програми з користувачем, тобто інтерфейсна частина алгоритму, є дуже важливою і повинна бути оформлена відповідним чином. Якщо ви не попіклуєтеся про коректне введення-виведення даних, то вам не зовсім буде зрозуміло, що «хоче» від вас комп'ютер на кроці виконання програми.

Тепер перейдемо до введення елементів двовимірного масиву.

```
write ('Задайте кількість рядків масиву:');
readln (n);
write ('Задайте кількість стовпчиків масиву:');
readln (m);
writeln ('Введіть елементи масиву:');
for i:=1 to n do
  for j:=1 to m do
    begin
      write ('A[',i,',',j,']=');
      readln (A[i,j])
    end;
```

В якості прикладу виведення елементів двовимірного масиву візьмемо варіант виведення його значень по рядках.

```
writeln ('Елементи результуючого масиву:');
for i:=1 to n do
  begin
    for j:=1 to m do
      write (A[i,j],' ');
    writeln
  end;
```

Використання вкладених циклів в наведеному прикладі слід розібрати. Зовнішній цикл містить два оператори: оператор внутрішнього циклу, який виводить в один рядок на екрані елементи поточного рядка j нашої прямокутної таблиці, і процедуру **writeln**. Це означає, що після виведення кожного рядка масиву,

здійснюватиметься перехід на новий рядок екрану. Таким чином одержимо таблицю результатів у звичному вигляді.

Після того, як ми опанували інтерфейсною частиною програми, можемо перейти до типових моментів обробки самого масиву. Найчастіше задачі, що передбачають роботу з таблицями, вимагають зміни значень самих заданих таблиць, або використання значень заданих табличних величин для обчислення інших таблиць.

Розглянемо ці два випадки.

Приклад 1. Нехай нам необхідно визначити елементи таблиці за правилом A[i,j]=i+j. Фрагмент програми буде виглядати так:

```
for i:=1 to n do
   for j:=1 to m do
      A[i,j]:=i+j;
```

Приклад 2. Вважатимемо, що нам задані два двовимірні масиви A[i,j] та B[i,j] розмірністю *n*m*, тобто обидві таблиці мають *n* рядків та *m* стовпчиків. Нам необхідно порахувати значення масиву C[i,j] такої ж розмірності, елементи якого є поелементною сумою заданих масивів. Тобто до кожного елементу таблиці *A* треба додати елемент таблиці *B*, розташований в ній на такому ж місці, і результат записати на таке ж місце в таблиці *C*. Повний варіант алгоритму у вигляді програми буде таким.

```
program suma;
var A,B,C: array[1..100,1..100] of real;
  n,m,i,j: integer;
begin
   write ('Задайте кількість рядків масивів:');
   readln (n);
   write ('Задайте кількість стовпчиків масивів:');
   readln (m);
   writeln ('Введіть елементи масиву A:');
   for i:=1 to n do
      for j:=1 to m do
        begin
          write ('A[',i,',',j,']=');
          readln (A[i,j])
        end;
    writeln ('Введіть елементи масиву B:');
   for i:=1 to n do
      for j:=1 to m do
        begin
          write ('B[',i,',',j,']=');
          readln (B[i,j])
        end;
   for i:=1 to n do
      for j:=1 to m do
        C[i,j]:=A[i,j]+B[i,j];
   writeln ('Елементи результуючого масиву C:');
   for i:=1 to n do
      begin
        for j:=1 to m do
          write (C[i,j],' ');
        writeln
      end;
end.
```


Алгоритми для роботи з одновимірними масивами аналогічні наведеним, але для їх обробки використовуються прості цикли.

Приклад 3. Розглянемо ще одну таку задачу. Нехай задана таблиця здійснення рейсів *N* автобусами протягом *M* днів. Рейси, що відбулися, позначаються в таблиці цифрою 1, а ті, що за деяких причин не відбулися, цифрою 0. Необхідно скласти програму, яка підраховує кількість рейсів, що не відбулися.

```
program voyage;
 var
   R : array[1..30,1..100] of byte;
   i, j, n,m, count  : integer;
begin
  writeln ('Задайте кількість автобусів:');
  repeat
    read(n)
  until (n>0) and (n<=30);
  writeln ('Задайте кількість днів:');
  repeat
    read(m)
  until (m>0) and (m<=100);
  writeln ('Задайте інформацію про здійснення рейсів:');
        {Введення початкових даних }
```

```
  for i:=1 to n do
    for j:=1 to m do
      begin
        writeln (i,' автобус, ',j,' день:');
        repeat
          read(R[i,j])
        until (R[i,j]=0) or (R[i,j]=1)
      end;
  count:=0;          {Підготовка змінної для накопичення суми}
  for i:=1 to n do              {Підрахунок кількості рейсів,}
    for j:=1 to m do            {що не відбулися }
      if R[i,j]=0 then count:=Inc(count);
  writeln ('Кількість нездійснених рейсів:',count:5);
  repeat until KeyPressed
end.
```

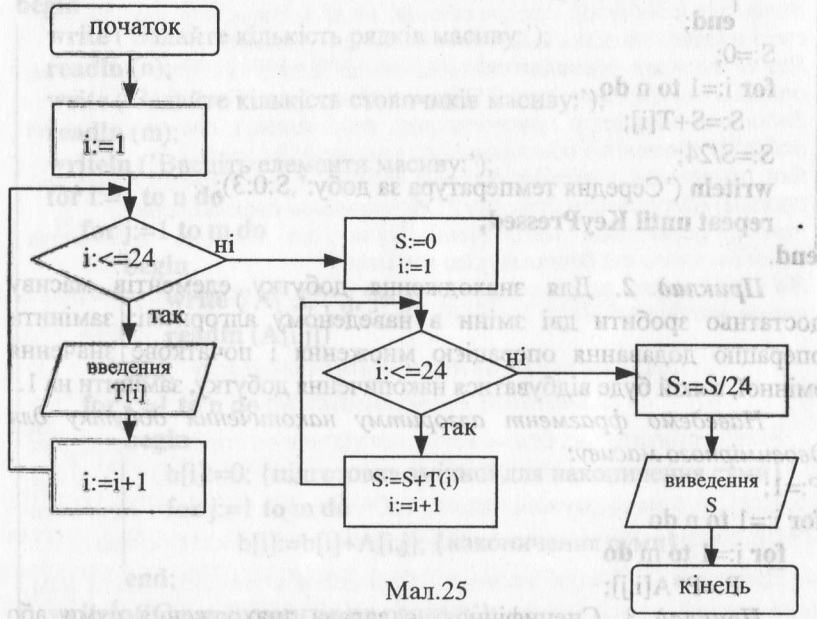
У цій програмі ми використали нову стандартну функцію Паскаля **Inc**(count). Її призначення – збільшення значення параметра на 1. Можливий ще такий варіант використання цієї функції: **Inc**(n,step). В цьому випадку збільшення параметра n буде відбуватися з кроком step. У Паскалі існує альтернативна процедура **Dec**, яка відповідно зменшує значення вказаного параметра.

Питання для самоконтролю:

1 *Запишіть фрагмент програми, що відображає введення початкових значень для елементів одновимірного масиву.*
2 *Запишіть фрагмент програми, що відображає виведення початкових значень для елементів одновимірного масиву.*
3 *Якими способами можна відділити значення елементів масиву при виведенні на екран монітора?*
4 *Запишіть фрагмент програми, що відображає введення початкових значень для елементів двовимірного масиву.*
5 *Запишіть фрагмент програми, що відображає виведення початкових значень для елементів двовимірного масиву.*
6 *Які дії над елементами табличних величин найчастіше зустрічаються у задачах?*
7 *Яким чином можна задавати початкові дані для елементів масивів?*
8 *Наведіть власний приклад алгоритму і запишіть програму для обробки елементів одновимірного масиву.*
9 *Наведіть власний приклад алгоритму і запишіть програму для обробки елементів двовимірного масиву.*

31. Алгоритми знаходження суми й добутку елементів таблиць

Приклад 1. Розглянемо схему алгоритму (мал.25), за допомогою якої можна визначити середнє арифметичне значення добової температури, оскільки саме для визначення середнього арифметичного значення ми не обійдемося без визначення суми



Мал.25

елементів масиву. А далі спробуємо за наведеною схемою алгоритму розробити Паскаль-програму.

Проаналізувавши схематичне зображення алгоритму, можна зробити висновок, що ми маємо справу з одновимірним масивом, що містить 24 елементи. Для підрахунку суми елементів масиву визначена змінна S, яка на початку набуває значення 0. Результатом виконання алгоритму є отримане значення S/24. Якщо навіть вважати, що початкові значення температур є цілими числами, то в результаті все одно буде отримане дійсне число.

А тепер програма:

```
program seredne;
var  T: array[1..24] of integer;
     i: integer;
```

```
    S: real;
begin
    writeln ('Введіть значення температур:');
    for i:=1 to 24 do
        begin
            write ('T[',i,']=');
            readln (T[i])
        end;
    S:=0;
    for i:=1 to n do
        S:=S+T[i];
    S:=S/24;
    writeln ('Середня температура за добу:',S:0:3);
    repeat until KeyPressed;
end.
```

Приклад 2. Для знаходження добутку елементів масиву достатньо зробити дві зміни в наведеному алгоритмі: замінити операцію додавання операцією множення і початкове значення змінної, в якій буде відбуватися накопичення добутку, замінити на 1.

Наведемо фрагмент алгоритму накопичення добутку для двовимірного масиву:

```
P:=1;
for i:=1 to n do
    for j:=1 to m do
        P:=P*A[i,j];
```

Приклад 3. Специфічною є задача знаходження суми або добутку не всіх елементів прямокутної таблиці, тобто двовимірного масиву, а кожного його рядка (або стовпчика) окремо. Розглянемо реалізацію цього алгоритму. Спочатку визначимо, скільки результатів ми отримаємо при такому підрахунку. Якщо в попередніх задачах результат був один – загальна сума, то в даному випадку результатів буде стільки, скільки в таблиці визначено рядків (або стовпчиків). Тому можна зробити висновок, що результатом виконання даного алгоритму буде одновимірний масив значень. Другий момент, з яким необхідно розібратися, перш ніж приступати до запису алгоритму у вигляді програми, це місце в алгоритмі, де буде відбуватись сумування. Уявивши собі, як би ми знаходили суму цих елементів таблиці вручну, можна дійти висновку, що дану

послідовність дій треба виконувати покроково при кожному перегляді рядка (стовпчика). Отож перейдемо до програми.

```
program suma;
    uses crt;
    var A: array[1..100,1..100] of real;
        b: array[1..100] of real;
        n,m,i,j: integer;
begin
    write ('Задайте кількість рядків масиву:');
    readln (n);
    write ('Задайте кількість стовпчиків масиву:');
    readln (m);
    writeln ('Введіть елементи масиву:');
    for i:=1 to n do
        for j:=1 to m do
            begin
                write ('A[',i,',',j,']=');
                readln (A[i,j])
            end;
        for i:=1 to n do
            begin
                b[i]:=0; {підготовка змінної для накопичення суми}
                for j:=1 to m do
                    b[i]:=b[i]+A[i,j]; {накопичення суми}
            end;
    writeln('Сума елементів по рядках:');
    for i:=1 to n do
        write(b[i],' ');
    repeat until KeyPressed
end.
```

Для визначення суми елементів по стовпчиках необхідно перегляд рядків і стовпчиків поміняти місцями:

```
for j:=1 to m do
  begin
    b[j]:=0; {підготовка змінної для накопичення суми}
    for i:=1 to n do
        b[j]:=b[j]+A[i,j]; {накопичення суми}
  end;
```

```
writeln('Сума елементів по рядках:');
for j:=1 to m do
    write(b[j],' ');
```

Зміни в алгоритмі підрахунку добутку елементів по рядках і стовпцях полягають у заміні операції «+» на операцію «*» та початкового значення змінної для накопичення результату «0» на «1».

Питання для самоконтролю:

1. *Запишіть оператор присвоювання, який виконує дію накопичення суми елементів одновимірного масиву A[i] у змінній S.*
2. *Яке початкове значення повинна мати змінна S перед початком операції накопичення суми? Обґрунтуйте свою відповідь.*
3. *Запишіть оператор присвоювання, який виконує дію накопичення добутку елементів одновимірного масиву A[i] у змінній P.*
4. *Яке початкове значення повинна мати змінна P перед початком операції накопичення добутку? Обґрунтуйте свою відповідь.*
5. *Чим відрізняється накопичення суми та добутку елементів одновимірного та двовимірного масивів?*
6. *Які особливості алгоритму підрахунку суми та добутку елементів рядків або стовпчиків у двовимірному масиві?*

32. Алгоритм пошуку елементів з деякою властивістю

Якщо спитати у будь-якого досвідченого програміста, які задачі найчастіше зустрічаються у його практиці, швидше за все він відповість, що це пошукові задачі, тобто пошук заданого елемента в деякій послідовності значень. Сформулюємо такі класичні задачі.

Приклад 1. Пошук заданого елемента в масиві. Нехай нам задано деякий масив значень $a_1, a_2, ..., a_n$ і значення величини x. Необхідно визначити, чи містить вказаний масив величину x, і якщо так, то на якому місці в масиві він знаходиться.

Алгоритм задачі базується на послідовній лінійній перевірці всіх елементів масиву $a1, a2, ..., a_n$ на співпадання з величиною x. Це можна зробити лише застосувавши алгоритмічну структуру повторення, яке повинно відбуватися стільки разів, допоки ми не знайдемо шуканий елемент.

Давайте спочатку розглянемо алгоритм-програму, а потім проаналізуємо його.

```
program search;
  uses crt;
  var a:          array[1..100] of real;
```



```
      x:    real;
      i, n: integer;
begin
    write ('Задайте кількість елементів масиву:');
    readln (n);
    writeln ('Задайте елементи масиву:');
    for i:=1 to n do
      readln (a[i]);
    write ('Задайте значення шуканої величини:');
    readln (x);
    i:=1;
    while (a[i]<>x) and (i<=n) do
      i:=i+1;
    if i<=n
      then
        writeln('Елемент ',x:5:2,' знаходиться в масиві на ',i,' місці');
      else
        writeln('Елемент ',x:5:2,' в масиві відсутній');
    repeat until KeyPressed
end.
```

В наведеному прикладі алгоритму варто зупинитися на умові (a[i]<>x) **and** (i<=n), що визначена в циклі з передумовою. Її слід читати так: ми продовжуємо виконувати цикл, якщо ще не знайдено шуканого елемента в масиві (a[i]<>x) та якщо ми ще не вийшли за межі масиву (i<=n). В самому циклі виконується лише одна операція переміщення на наступний елемент масиву: i:=i+1. Коли ми вийдемо із циклу, необхідно визначити причину завершення циклу. Їх може бути дві: або знайдено шуканий елемент в масиві (a[i]=x), або елемента немає, і ми вийшли за межі масиву (i>n). Саме ці два можливі випадки і перевіряє оператор розгалуження після завершення циклу та визначає, яку результуючу інформацію необхідно вивести на екран монітора: шуканий елемент знайдено, і це сталося при значенні індекса масиву **i**, або він в масиві відсутній.

Приклад 2. Пошук мінімального (максимального) елемента в масиві. Нехай задано деякий масив $a_1, a_2, ..., a_n$. Необхідно знайти мінімальний елемент цього масиву (*min*).

Алгоритм пошуку найменшого елемента будуватимемо за таким сценарієм.

Від початку до кінця масиву відкриватимемо послідовно по одному елементу і на кожному кроці визначатимемо поточний найменший елемент. На першому кроці найменшим буде перший елемент a_1. На кожному наступному кроці порівнюватимемо новий відкритий елемент a_i з тим мінімумом, який ми вже маємо. Якщо новий елемент менший від того мінімального, що визначений на попередньому кроці, то запам'ятаємо його, інакше залишимо старий результат. Таким чином, дійшовши до кінця масиву, ми визначимо мінімальний елемент всього масиву.

Програма, записана на основі наведеного алгоритму буде мати такий вигляд:

```
program minimum;
  uses crt;
  var a:  array[1..100] of real;
    min:  real;
    i, n, k:    integer;
begin
    write ('Задайте кількість елементів масиву:');
    readln (n);
    writeln ('Задайте елементи масиву:');
    for i:=1 to n do
        readln (a[i]);
    min:=a[1]; k:=1;
    for i:=2 to n do
        if a[i]<min then
          begin
            min:=a[i]; k:=i
          end;
    writeln('Мінімальний елемент ',min);
    writeln('знаходиться в масиві на ',k,' місці');
    repeat until KeyPressed
end.
```

Запропонована програма не тільки знаходить найменший елемент, але ще й визначає, на якому місці в масиві він знаходиться.

Проаналізуємо декілька моментів стосовно даного алгоритму, розглянувши такі варіації на тему «Пошук оптимального значення».

- Наведений приклад алгоритму визначає перший мінімальний елемент в масиві, хоча в ньому може бути і декілька таких елементів. Наприклад, якщо розглянути масив 1,2,2,1,3,1, то мінімальним елементом буде перша одиниця. Весь секрет ховається в умові a[i]<min. Адже новий елемент масиву буде запам'ятовуватися як найменший лише тоді, коли він менший за попередній.
- Якщо в алгоритмі поміняти умову a[i]<min на умову a[i]<=min, то визначатиметься останній найменший елемент, оскільки новий елемент масиву буде запам'ятовуватися ще й тоді, коли він рівний попередньому.

Цього досить, щоб розглянутий алгоритм виконував все, про що йшла мова вище. Щоб знайти найбільший елемент, потрібно умову a[i]<min поміняти на a[i]>max.

Питання для самоконтролю:

1 *Для чого використовуються пошукові алгоритми?*
2 *Запишіть програму пошуку заданого елемента в одновимірному масиві та проаналізуйте її.*
3 *У чому полягає ідея пошуку мінімального елемента в масиві?*
4 *Запишіть програму пошуку мінімального елемента в одновимірному масиві та проаналізуйте її.*
5 *Чим відрізняються алгоритми пошуку мінімального та максимального значення в масиві?*
6 *Як визначити порядковий номер першого мінімального (максимального) значення в масиві?*
7 *Як визначити порядковий номер останнього мінімального (максимального) значення в масиві?*

33. Практична робота «Опрацювання табличних величин»[6]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм з табличними величинами.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*

[6] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Опрацювання табличних величин»

2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму або його фрагментів,побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму та побудувати його схему.*
6) *Визначити типи табличних величин (лінійні, прямокутні), які необхідно застосувати для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання послідовних та вкладених повторень, застосування розгалужень для обробки масивів.*
9) *Записати алгоритм мовою програмування.*
10) *Набрати текст програми, використовуючи середовище програмування, та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*
12) *Реалізувати інші варіанти використання повторень і розгалужень для виконання сформульованої задачі та проаналізувати їх щодо кількості виконуваних дій для різних початкових даних.*

34. Алгоритми впорядкування табличних величин

Нехай задано деякий масив значень $a1$, $a2$, ..., a_n. Необхідно перетворити його таким чином, щоб виконувалося співвідношення $a'_1 <= a'_2 <= ... <= a'_n$.

Алгоритми впорядкування елементів в лінійних табличних величинах поділяються на три типи в залежності від застосованого методу. Існує таких три методи впорядкування:

- метод вибору;
- метод обміну;
- метод включення.

Розглянемо перший метод – *метод вибору.*

Якщо спробувати навести порядок у картках, на яких написані деякі числа, то, напевно, дійшли б такого висновку. Спочатку необхідно знайти мінімальний елемент у всьому заданому масиві і поміняти його з елементом, який поки що стоїть на першому місці тому, що саме на цьому місці повинен стояти знайдений мінімальний елемент. Тепер вже перший елемент знаходиться на своєму місці, і ми можемо його закрити, не розглядаючи далі. Наш масив зменшився на один елемент (перший). З новим масивом виконаємо ті ж самі дії: знайдемо в ньому мінімальний елемент і поміняємо його місцями з першим відкритим елементом (насправді він є другим елементом в заданому масиві). Таким чином будемо продовжувати до того часу, поки на останньому кроці не залишаться два останні елементи. Якщо необхідно, поміняємо і їх місцями. Таким чином ми з вами визначилися щодо алгоритму впорядкування послідовності значень методом вибору. Цей алгоритм у вигляді програми буде таким:

```
program order;
 uses crt;
 var a:          array[1..100] of real;
     min, c:     real;
     i, j, n, k: integer;
 begin
     writeln ('Задайте кількість елементів масиву:');
     read (n);
     writeln ('Задайте елементи масиву:');
     for i:=1 to n do
        readln (a[i]);
     for i:=1 to n-1 do
        begin
           min:=a[i]; k:=i;
           for j:=i+1 to n do
              if a[j]<min then
                 begin
                    min:=a[j]; k:=j
                 end
           c:=a[i]; a[i]:=a[k]; a[k]:=c;
        end;
```

```
writeln ('Упорядкований масив:');
for i:=1 to n do
    write (a[i],' ');
repeat until KeyPressed
end.
```

Для того, аби описаний алгоритм виконував упорядкування масиву за спаданням, необхідно логічний вираз a[j]<min замінити на a[j]>min, замінивши також при бажанні ідентифікатор min на ідентифікатор max.

Алгоритм, що застосовує *метод обміну* для впорядкування одновимірного масиву, ще носить назву «бульбашкового». Це пояснюється тим, що на кожному кроці алгоритму розглядаємо з кінця до початку два сусідніх елемента і міняємо їх місцями у випадку, якщо правий елемент менший за лівий. При цьому відбувається «виштовхування» найлегшого (найменшого) елемента в початок. Оскільки початок і кінець програми аналогічні попередньому варіанту, розглянемо лише фрагмент безпосереднього упорядкування.

```
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;
```

Наприклад, розглянемо масив 44, 55, 12, 42, 94, 18, 6, 67. На першому кроці, розглядаючи весь масив з кінця до початку, будемо переміщувати найменший елемент на перше місце. Це буде число 06.

В результаті виконання внутрішнього циклу отримаємо перетворений масив 6, 44, 55, 12, 42, 94, 18, 67. Тепер вступає в роботу зовнішній цикл. Він «скорочує» наш масив, оскільки найменший елемент вже стоїть на своєму місці. Розглядаємо масив 44, 55, 12, 42, 94, 18, 67. В результаті роботи внутрішнього циклу найменший його елемент стане на своє місце (в початковому масиві це буде друге місце). Отримаємо такий проміжний результат сортування: 18, 44, 55, 12, 42, 94, 67. Цей процес буде продовжуватися, поки на останньому кроці не буде здійснено при необхідності обмін двох найбільших елементів.

І на завершення знайомства з алгоритмами упорядкування – *метод включення.*

Нехай задана послідовність a1, a2, ..., an. Відомо, що на i-му кроці a1, a2, ..., ai-1 –впорядкована частина послідовності. Ідея алгоритму цього методу полягає в тому, що для кожного i=2,3,...,n береться елемент a[i] і ставиться у впорядковану частину послідовності, що йому передує, на своє місце. Для реалізації даного алгоритму необхідно розширити проміжок зміни індексів масиву: var a: array[0..100] of real;. У вигляді фрагмента програми цей алгоритм буде таким:

```
for i:=2 to n do
  begin
    x:=a[i]; a[0]:=x; j:=i;
    while (x<a[j-1]) do
      begin
        a[j]:=a[j-1];
        j:=j-1
      end;
    a[j]:=x
  end;
```

Розглянемо алгоритм на тому самому прикладі масиву: 44, 55, 12, 42, 94, 18, 6, 67. Зовнішній цикл на кожному кроці буде «зсувати» початок масиву, що розглядається у внутрішньому циклі. Спочатку це буде весь масив, потім, починаючи з другого елемента, далі – з третього і т.д. Нас зараз буде цікавити, що робить внутрішній цикл, як відбувається визначення місця для найменшого елемента на даному кроці. Мабуть, ви вже помітили, що в цьому прикладі алгоритму внутрішній цикл організований як цикл з передумовою. Справа в тім, що якщо вже знайдено місцезнаходження елемента, що ставиться на своє місце, то немає смислу продовжувати перегляд масиву далі.

Отож, проаналізуємо алгоритм. Для кращого розуміння пояснення роботи алгоритму варто вручну виконувати покроково всі наступні дії.

На першому кроці зовнішнього циклу встановлюються такі значення змінних: i=2, x:=a[2] (x=55), a[0]=55, j:=2. Призначення штучно введеного елемента масиву a[0] ми побачимо пізніше.

Умова внутрішнього циклу не справджується (55<44), тому все лишається на своїх місцях.

На другому кроці зовнішнього циклу i=3, x:=a[3] (x=12), a[0]=12, j:=3. Починаємо виконувати внутрішній цикл. На його першому кроці умова справджується – 12<55. Заходимо всередину внутрішнього циклу і виконуємо присвоєння: на третє місце в масиві ставимо число 55 (масив: 44, 55, 55, 42, 94, 18, 6, 67). Чи можна це робити, чи не пошкодимо ми масив? Ні, оскільки число, що було на третьому місці, тобто 12, ми запам'ятали у змінній x перед початком внутрішнього циклу. Змінюємо параметр внутрішнього циклу j:=j-1 (j=2) і переходимо до наступного його кроку. Знову перевіряємо умову циклу. Тепер вона вже має вигляд 12<44. Умова справджується, і тому знову виконуємо внутрішній цикл, на друге місце в масиві ставимо число 44 (масив: 44, 44, 55, 42, 94, 18, 6, 67). На завершення цього кроку внутрішнього циклу j=1. При поверненні на початок внутрішнього циклу вступає в роботу a[0], оскільки перевіряється умова x<a[0] (12<12). Ця умова не виконується, і внутрішній цикл припиняє свою роботу. Але після його завершення у зовнішньому циклі є ще один оператор a[j]:=x, який при поточних значеннях змінних має вигляд a[1]:=12. Тепер все стало на свої місця і масив має остаточний вигляд: 12, 44, 55, 42, 94, 18, 6, 67.

Не варто розглядати наступні кроки виконання зовнішнього циклу, оскільки вони аналогічні. Можна зробити такий висновок:

- на кожному кроці зовнішнього циклу поточний елемент масиву, рухаючись вліво на початок масиву, «шукає» своє місце в уже відсортованій лівій частині;
- рух поточного елемента вліво відбувається до того моменту, поки не виконається умова 'поточний елемент'>='елемент лівої частини масиву', тобто цей рух не завжди сягає першого елемента заданого масиву.

Насамкінець, проаналізуємо наведені методи впорядкування одновимірних масивів. Критерієм оцінки визначимо кількість виконуваних дій (кроків циклу) для отримання результату. З цієї точки зору перші два методи мало чим відрізняються один від одного. Обидва використовують два вкладених цикла типу **for…to…do**. Тому кількість виконуваних повторень при виконанні цих циклів залежить від кількості елементів. Третій алгоритм суттєво відрізняється від перших двох, оскільки внутрішній цикл є циклом з передумовою **while…do**. Кількість кроків цього циклу залежить від виконання умови x<a[j-1]. А це означає , що він завершиться, коли ми знайдемо місце елемента x в уже впорядкованій лівій частині масиву.

Тому метод включення варто використовувати, коли нам треба впорядкувати майже впорядкований масив.

Питання для самоконтролю:

1 *Які існують методи впорядкування масивів?*
2 *Скільки циклів необхідно організувати в програмі для впорядкування значень елементів одновимірного масиву?*
3 *Які цикли використовуються в програмі упорядкування значень елементів масиву – вкладені чи послідовні? Поясніть свою відповідь.*
4 *Запишіть програму, що реалізує алгоритм впорядкування елементів одновимірного масиву методом вибору.*
5 *Яким чином використовується пошук мінімального (або максимального) елемента масиву для його упорядкування методом вибору?*
6 *Чим відрізняється алгоритм впорядкування значень елементів масиву за зростанням та за спаданням методом вибору?*
7 *Запишіть програму, що реалізує алгоритм впорядкування елементів одновимірного масиву методом обміну.*
8 *Поясніть алгоритм впорядкування елементів одновимірного масиву методом обміну.*
9 *Запишіть програму, що реалізує алгоритм впорядкування елементів одновимірного масиву методом включення.*
10 *Наведіть приклад одновимірного масиву і поясніть покроково роботу алгоритму, його впорядкування методом включення.*

35. Практична робота «Впорядкування табличних величин»[7]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм впорядкування табличних величин.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*

[7] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Впорядкування табличних величин»

3) *Визначити метод впорядкування табличної величини, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму.*
6) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
7) *Визначити коректність використання послідовних та вкладених повторень, застосування розгалужень для впорядкування масивів.*
8) *Обґрунтувати вибір метода впорядкування табличної величини відповідно до умови задачі.*
9) *Записати алгоритм мовою програмування.*
10) *Набрати текст програми, використовуючи середовище програмування, та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*
12) *Реалізувати інші варіанти використання методів впорядкування масивів та проаналізувати їх щодо кількості виконуваних дій для різних початкових даних.*

36. Тематична атестація з теми «Табличні величини»

Для виконання тематичної атестації необхідно:

знати

- поняття табличних величин, їх опис мовою програмування Паскаль, поняття індексу масиву;
- означення одновимірних та двовимірних масивів та відмінність між ними;
- способи введення та виведення елементів масиву;
- принципи обробки елементів масиву;
- відмінність роботи з одновимірними та двовимірними масивами;
- алгоритми знаходження суми та добутку елементів одновимірного та двовимірного масивів;
- алгоритми пошуку елементів з деякою властивістю та їх кількість;

- методи впорядкування елементів масиву;

вміти

- описувати мовою програмування одновимірні та двовимірні масиви;
- розв'язувати задачі з використанням введення, обробки та виведення елементів масиву;
- використовувати різні методи впорядкування елементів одновимірного масиву;
- аналізувати алгоритми та їх реалізацію у вигляді програми з використанням масивів.

Рядкові величини

37. Вказівки і функції опрацювання рядкових величин

З рядковими величинами ми вже мали справу. Тепер давайте подивимося на них іншими очима. Адже це масиви символів. І дійсно, замість запису str: **string** можна записати str: **array**[1..255] **of char**. Найочевидніша відмінність в цих записах полягає в тому, що в першому випадку ви зможете ввести текст одним рядком, а в другому – лише по одній літері.

Давайте заглибимося у світ рядкових величин і дізнаємося багато корисного. Наприклад, якщо вам наперед відома приблизна довжина символьного рядка, який буде введений користувачем під час виконання програми, то можна тип string дещо обмежити:

string[n], де *n* – будь-яке число в межах 1..255.

В цьому разі під таку змінну буде відведено стільки байтів, якою є довжина описаної змінної.

Зауважте, що тип string є простим типом, а вже тип **string**[n] – структурованим.

Насправді рядкові величини мають на один символ більше, ніж для них відведено байтів, тобто максимально 256 символів. Весь секрет криється у символі з номером 0 (str[0]). Там міститься інформація про справжню кількість введених символів рядкової величини.

Виконайте таку операцію **write** (**ord**(str[0]))

і отримаєте інформацію про те, скільки символів в рядку було введено.

До окремих символів рядкової величини можна дістатися так само, як і до елементів одновимірного масиву, вказавши порядковий номер необхідного символу. Наприклад, якщо була виконана операція str:='алгоритм', то str[4]='о'.

Визначимо операції над рядковими величинами.

Зчеплення. Ця операція дозволяє поєднувати декілька рядкових величин і позначається символом «+». Наведемо приклади:

str1:='алгоритм'; str2:='найкращий';
str1+' '+str2 ⇨ 'алгоритм найкращий';
str2+' '+str1 ⇨ 'найкращий алгоритм'.

Порівняння. Оскільки всі символи кодуються своїми порядковими номерами, які ще називають ASCII-кодами, то рядки можна порівнювати. При цьому будуть спочатку порівнюватися перші символи рядків, потім другі і т.д.

Наприклад:

'Алгоритм' < 'алгоритм', оскільки спочатку йдуть великі літери, а потім малі;

'алгоритм' > 'алго', оскільки 'р'>'': залишки рядків та їх довжини не грають ніякої ролі, код будь-якого символу завжди більше коду «порожнього місця» ;

'алгоритм' <> 'alhoritm', оскільки ASCII-коди латинських букв та букв кирилиці різні;

'алг' ='алг'.

Розглянемо функції та процедури, які існують у Паскалі для роботи з рядковими величинами.

1. Про довжину рядкової величини вам допоможе дізнатися стандартна функція Паскаля, що має такий вигляд:

Length(S), де *S* – рядкова величина.

Результатом виконання цієї функції буде ціле число типу **byte**. Наприклад, **Length**('alhoritm')=8; **Length**(S)=8 при S='alhoritm'.

2. Для визначення місцезнаходження шуканого фрагменту рядкової величини можна скористатися функцією

pos (S_find, S), де S_find – шуканий підрядок, S – рядок.

Результат виконання цієї функції – величина типу **byte**. Якщо шуканого підрядка в рядку не існує, то функція **pos** повертає

значення 0. Наприклад, **pos**('a',S)=1 при S='alhoritm'; **pos**(S1,S)=4 при S='My program', S1='program'.

3. Наступна функція дасть той самий ефект, що й операція зчеплення:

concat(<список рядкових величин>),

де список рядкових величин – це величини, значення яких будуть зчеплені. Результат виконання – змінна типу **string**. Наприклад, **concat**(S,' ',S1)='My program' при S='My', S1='program'.

4. Якщо вам необхідно виділити деякий фрагмент рядкової величини, скористайтеся функцією

copy(S, Start, Len),

де S – рядкова величина, з якої виділяється фрагмент, Start – ціле значення, що визначає початкову позицію фрагменту, який виділяється, Len – ціле значення, що визначає кількість символів фрагменту. Результатом виконання цієї процедури є величина типу **string**. Будьте уважні: якщо виділяєте фрагмент лише в один символ, то результатом буде значення не типу **char**, а все одно **string**! Приклад: **copy**(S,4,7)='program' при S='My program'.

5. Переходимо до знайомства з процедурами. Першою буде процедура, що дозволяє вилучити деякий фрагмент з рядкової величини.

delete(S, Start, Len),

де S – рядкова величина, з якої вилучається фрагмент, Start – порядковий номер першого символа фрагменту, що вилучається, Len – кількість символів фрагменту, що вилучається. Наведемо приклад: при виконанні процедури **delete**(S,4,4) при S='My new program' буде отримано результат S='My program'.

6. Аналогічно можна вставити фрагмент у рядкову величину. Для цього існує процедура

insert(S, S_new, Start),

де S – рядкова величина, фрагмент, що вставляється, S_new – рядкова величина, в яку вставляється фрагмент і де отримується результат вставлення, Start – порядковий номер символа в заданій рядковій величині, після якого вставляється фрагмент. Приклад: виконання процедури **insert**(S,S1,3) при S='new ', S1='My program' дасть результат S1='My new program'; виконання процедури **insert**('new ',S1,3) при S1='My program' дасть результат S1='My new program'.

7. Досить часто зустрічається ситуація, коли рядкові величини мають числовий вигляд і виникає необхідність перетворення їх у числовий тип. В цьому вам допоможе процедура

val (S, V, ErrCode),

де S – рядкова величина, V – величина числового типу, який ви очікуєте одержати, ErrCode – змінна, значення якої дорівнює коду виконання процедури (ErrCode=0 – перетворення відбулося успішно). У якості прикладу можна навести такий: після виконання процедури **val** (S,V,ErrCode) при S='1234' буде отримано V=1234, ErrCode=0.

8. Зворотна процедура до **val** виглядає таким чином

str(V, S),

де V – величина числового типу, значення якої перетворюється у рядковий вигляд, S – результат перетворення рядкового типу. Приклад: виконання процедури **str**(1234, S) призведе до отримання результату S='1234'; виконання процедури **str**(V,S) при V=12.34 призведе до отримання результату S='1.2340000000E+01'

Питання для самоконтролю:

1 *Чим пояснити те, що рядкові величини подібні до масивів?*
2 *Яка максимальна довжина символьних масивів?*
3 *Де знаходиться інформація про кількість заданих елементів символьного рядка?*
4 *Які операції дозволені над рядковими величинами?*
5 *Назвіть функції для роботи з рядковими величинами та призначення їх параметрів.*
6 *Назвіть процедури для роботи з рядковими величинами та призначення їх параметрів.*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

При складанні та реалізації алгоритмів з рядковими величинами ви можете зустрітись з тими ж самими помилками, що і при роботі з масивами. Тому звертайтесь до аналізу помилок у попередньому розділі.

Якщо результат функції **copy** присвоюється змінній типу **char**, то буде виведено повідомлення про помилку щодо неспівпадання типів

Error 26: Type mismatch.

38. Складання алгоритмів з використанням рядкових величин

Найтиповішим прикладом використання рядкових величин є аналіз окремих символів в них з метою їх заміни, вилучення, вставлення тощо.

Приклад 1. Нехай нам задано текст, в якому необхідно всі російські букві 'ы' замінити на українські 'і'. Суть алгоритму буде полягати в пошуку в тексті чергової букви 'ы' і заміни її на цьому ж місці на букву 'і'. Оскільки до символів в тексті ми можемо звертатися, як до елементів масиву, то алгоритм-програма буде виглядати таким чином:

```
program change;
  uses crt;
  var
     S   : string;
     i   : integer;
  begin
     write ('Введіть текст: ');
     readln(S);
     for i:=1 to length(S) do
          if S[i]='ы' then S[I]:='і';          {заміна символу }
     writeln ('Замінений текст: ',S);
     repeat until KeyPressed
  end.
```

Слід нагадати, що значення типу **String** вводяться за допомогою процедури **Readln**, а не **Read**. Більше того, за один раз може бути введений лише один рядок.

Для вилучення і вставлення фрагментів тексту використовуються відповідні процедури **delete**(S, Start, Len) та **insert**(S, S_new, Start).

Запишемо той самий алгоритм заміни букв в тексті з використанням цих процедур:

```
for i:=1 to length(S) do
   if S[i]='ы'
     then
        begin
           delete(S, i, 1);
```

```
            insert(S, 'i', i)
        end;
```

Зауважте, що цікава ситуація зустрілася в цьому фрагменті програми: 'i' означає конкретний символ, а **i** – ім'я змінної!

Зрозуміло, що перший варіант алгоритму має переваги, оскільки заміна проводилася «за місцем», а в другому варіанті використовувались виклики процедур, що забирає зайвий час на виконання програми. Процедури **insert** та **delete** варто використовувати тоді, коли змінюється довжина тексту. Наприклад, коли треба замінити '...' на '.'.

Приклад 2. Розглянемо приклад алгоритму, який дозволяє записати задане слово навпаки (наприклад, 'алгоритм' ⇨'мтирогла').

Спочатку наведемо текст програми, що реалізує цей алгоритм, а потім проаналізуємо його.

```
program on_the_contrary;
  uses crt;
  var
    S,S_new      : string;
    i            : integer;
begin
  writeln ('Введіть слово:');
  readln(S);
  S_new:= '';
  for i:=1 to length(S) do
    S_new:=S[i]+S_new;          {дописування символу ліворуч }
  writeln ('Слово навпаки: ',S_new);
  repeat until KeyPressed
end.
```

Даний алгоритм є своєрідним прикладом алгоритму накопичення суми. Але у випадку роботи з рядковими величинами накопичення суми перетворюється у «дописування» символів в кінець рядка. Ви пам'ятаєте, що для накопичення суми змінна, де відбувається сумування, повинна мати початкове значення 0. Якщо перефразувати це твердження, сказавши, що перед початком сумування наша скринька повинна бути «порожньою», тобто там нічого не повинно бути, то в термінах рядкових величин це означатиме, що на початку формування рядкової величини відповідна змінна не повинна містити ніякого тексту. Оператор присвоювання S_new:= '' виконує саме таку роль.

До речі, якщо оператор S_new:=S[i]+S_new замінити на оператор S_new:=S_new+ S[i], то в результаті виконання програми заданий текст не зміниться.

Приклад 3. Варто розглянути ще один приклад роботи з рядковими величинами. На цей раз це буде масив рядкових величин.

Отже, розглянемо алгоритм-програму, яка за введеним списком рядків виводить їх в упорядкованому за алфавітом вигляді.

```
program Alphabetize;
  uses crt;
  var
    S       : array[1..100] of string;
    str     : string;
    i,j,n   : integer;
  begin
    write ('Введіть кількість слів: ');
    readln (n);
    writeln ('Введіть слова:');
    for i:=1 to n do
      readln(S[i]);
    for i:=1 to n do
      begin
        k:=i;
        for j:=2 to n do
          if (S[j]<S[i]) then
            begin
              str:=S[j];
              k:=j
            end;
        str:=S[i]; S[i]:=S[j]; S[j]:=str;
      end;
    writeln ('Список слів за алфавітом:');
    for i:=1 to n do
      writeln(S[i]);
    repeat until KeyPressed
  end.
```

На прикладі цього алгоритму ми бачимо, що упорядкування можна застосовувати не лише для числових величин.

Питання для самоконтролю:

1 *В чому полягає особливість використання для рядкових величин стандартних процедур введення?*
2 *В чому особливість заміни символів у заданому тексті?*
3 *Чим відрізняються операції S_new:=S[i]+S_new та S_new:=S_new+ S[i]? Обґрунтуйте свою відповідь.*
4 *Наведіть власний приклад програми для роботи з рядковими величинами.*

39. Практична робота «Опрацювання рядкових величин»[8]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм з використанням рядкових величин.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму або його фрагментів, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму.*
6) *Визначити операції над рядковими величинами, які необхідно застосувати для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання послідовних та вкладених розгалужень і повторень.*
9) *Записати алгоритм мовою програмування.*

[8] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Опрацювання рядкових величин»

10) *Набрати текст програми, використовуючи середовище програмування, та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*
12) *Реалізувати інші варіанти використання повторень, розгалужень, операцій над рядковими величинами, стандартних процедур і функцій для виконання сформульованої задачі та проаналізувати їх щодо кількості виконуваних дій.*

40. Тематична атестація з теми «Рядкові величини»

Для виконання тематичної атестації необхідно:

знати

- опис рядкових змінних мовою програмування;
- функції та процедури обробки рядкових величин;
- основні алгоритми обробки рядкових величин;

вміти

- описувати мовою програмування рядкові величини;
- розв'язувати задачі з використанням введення, обробки та виведення рядкових величин;
- аналізувати алгоритми та їх реалізацію у вигляді програми з використанням рядкових величин.

Звернення до процедур і функцій

41. Поняття основного та допоміжного алгоритму. Підпрограми

Локальні та глобальні змінні.
Формальні та фактичні параметри

Попередньо з процедурами та функціями ми мали справу, і не раз. Відмінність лише в тому, що відомі нам процедури та функції написав хтось інший, а в цьому розділі ви можете навчитися писати їх самі.

Кожна процедура або функція, тобто допоміжний алгоритм, це є «держава в державі», міні-програма в програмі. В ній діють всі закони, що притаманні будь-якій програмі. Структура процедур та функцій абсолютно така сама, як і всієї програми, тобто в них можуть бути свої розділи констант, міток, типів, змінних, процедур та

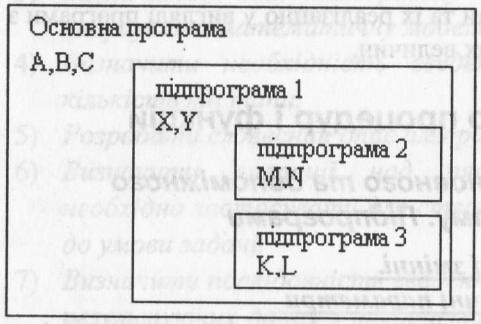
функцій. Але цікаво: все те, що описано в процедурах та функціях, доступно лише цим процедурам та функціям і не поширюється на всю програму. Логічно ці можливості назвати локальними.

А от всі можливості, що передбачені в основній програмі, доступні як основній програмі, так і всім процедурам та функціям, що в ній створені. Отже, тому вони називаються глобальними.

Все сказане вище поширюється в першу чергу на змінні.

***Локальними** називаються змінні, що описані в допоміжних алгоритмах. **Глобальними** називаються змінні, що описані в основному алгоритмі.*

Цікавий сам процес виконання процедур та функцій. Всім змінним, що описані в них, місце в пам'яті відводиться тільки під час виконання цих процедур або функцій. Після того, як процедура або функція відпрацює, змінні знищуються, тобто пам'ять від них вивільняється. Це наводить на думку, що в різних процедурах та функціях однієї і тієї ж програми, та й у самій цій програмі можуть використовуватися однакові ідентифікатори, і змінні, що їм відповідають, не будуть заважати одні одним.



Мал. 26

Схематично глобальні та локальні змінні зображені на мал.26.

Ви вже знаєте, що для того, аби викликати на виконання більшість процедур чи функцій, необхідно задати певні параметри. Цілком зрозуміло, що кожна з цих процедур та функцій повинна якимось чином «прийняти» ці значення, тобто необхідно передбачити їх одержання.

Параметри, які передаються у процедуру чи функцію при її виклику, називаються фактичними.

Параметри, що вказуються в заголовку процедури або функції і які при її виконанні набувають значень фактичних параметрів, називаються формальними.

Кількість фактичних і формальних параметрів при роботі з однією і тією ж самою процедурою чи функцією повинна бути однаковою. Окрім цього, повинні співпадати типи цих параметрів та їх зміст.

Типи підпрограм

Часто деяку послідовність дій слід повторити в декількох місцях програми. Щоб не витрачати зусиль на їх копіювання, в багатьох мовах програмування передбачені засоби для організації підпрограм. Таким чином програміст отримує можливість присвоїти послідовності операторів деяке ім'я і використовувати це ім'я в якості скороченого запису в тих місцях, де зустрічається відповідна послідовність дій.

Підпрограма – це засіб скорочення тексту програми та підвищення її структурованості.

Підпрограма є одним із багатьох фундаментальних інструментів, який впливає на стиль, якість, надійність розробки програми. Використання підпрограм значно підвищує зрозумілість всієї програми, особливо, якщо її текст складний і громіздкий. Подекуди рекомендується організовувати підпрограму навіть тоді, коли вона викликається лише один раз. Таким чином підпрограма є широко використовуваним засобом абстрагування: її можна розглядати як «чорний ящик», що забезпечує виконання деякої абстрактної функції. Користувача в цьому випадку не цікавить, ***як*** здійснюється виконання. Для нього важливим є лише те, ***що*** (які дії) виконує ця підпрограма.

Паскаль відноситься до тих мов програмування, в яких розрізняють два типи підпрограм – функції та процедури. Цей факт притаманний далеко не всім мовам. Наприклад, мова програмування Сі не розрізняє функції та процедури.

Саме про відмінність функцій та процедур і особливості їх використання у Паскалі ми поговоримо далі.

Опис підпрограм. Звернення до підпрограм

Функції характерні тим, що після їх виконання одержується лише один результат.

Загальний вигляд функції:

function *<ім'я функції> (<список формальних параметрів>):*
<тип результату>; .

<описова частина >

begin

<виконувана частина або тіло функції>

end;

Оскільки функція є самостійною програмною одиницею в межах основної програми, то всі змінні, які з'являються в ній, повинні бути описані в цій функції. І починається все з опису формальних параметрів – для кожного з них повинен бути вказаний тип. Описова частина може містити розділ змінних, необхідність в яких з'явилася при створенні даної функції (наприклад, лічильник циклу) і наявність яких у цій функції не чинить ніякого впливу на основну програму. Те ж саме стосується і всіх інших розділів.

Одне, але дуже значне, обмеження на типи параметрів, які ми можемо передавати у функцію (а надалі і в процедуру) – вони можуть бути лише змінними простого типу. Тобто поки що масиви та змінні типу string[n], які є структурованими, ми не використовуватимемо.

Яким же чином результат обчислення функції потрапить в основну програму? Для цього функція повинна містити оператор присвоювання, який і виконає пересилання результату в основну програму. Він має такий загальний вигляд:

<ім'я функції>:=<вираз>.

Зауваження! Ім'я функції може використовуватися лише для присвоювання йому деякого значення і не може брати участь в обчисленнях. Тобто запис

my_fun:=my_fun+1

буде сприйнятий як помилковий. Справа в тім, що використання імені функції у деякому виразі розцінюється як спроба повторного звернення до цієї функції. А як же бути в нашому випадку? Просто заведіть ще одну локальну змінну, обчисліть в ній необхідний результат, а потім за допомогою оператора присвоювання передайте його імені функції.

Загальний вигляд процедури:

procedure *<ім'я процедури>(<список формальних параметрів>);*

<описова частина >

begin

<виконувана частина або тіло процедури>

end;

Необхідно зразу ж зазначити, що в заголовку процедури не вказується її тип. А це тому, що за допомогою процедур ми з вами зможемо одразу знаходити і передавати в основну програму декілька обчислених значень. *Це і є основною відмінністю процедур від функцій.*

Для того, щоб зрозуміти, яким чином можна повернути результати з процедури в основну програму, введемо два нових поняття.

Параметри-значення. Простіше за все почати з прикладу.

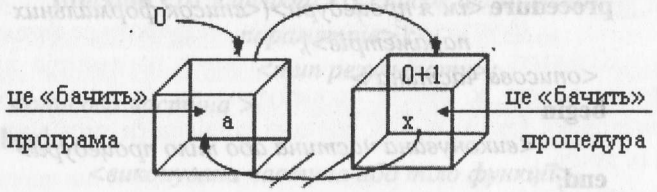
```
.....
procedure proba(x,y: integer);
  begin
    x:=x+1; y:=y+1
  end;
.....
begin
.....
  a:=0; b:=0;
  proba(a,b);
  writeln(a,b);
.....
```

Ця програма надрукує два числа: 0 0.

Розберемося детальніше в цій ситуації.

Під час виконання процедури параметрам-значенням відводиться нове місце в пам'яті, тобто створюється їх копія, куди передаються відповідні значення фактичних параметрів, над ними виконуються дії, вказані в процедурі, і після завершення роботи процедури всі ці змінні знищуються, а їх значення відповідно втрачаються. Тобто, які значення для a та b були задані перед зверненням до процедури, такими вони і залишилися після її виконання.

Схематично це виглядає так, як на мал.27.



Мал.27

Зрозуміло, що параметрами-значеннями варто користуватися у тих випадках, коли передаються у процедуру величини, які будуть брати участь в ній у деяких обчисленнях і які після повернення в основну програму повинні зберегти свої початкові значення.

І ще одна важлива деталь.

При зверненні до процедури параметрам-значенням можна передавати сталі величини.

Наприклад, (proba(2,5)). Чому це так важливо, ми зараз побачимо.

Параметри-змінні. Параметри-змінні на відміну від параметрів-значень після виконання процедури повертають ті значення, які вони одержали в процедурі. Справа в тім, що при зверненні до процедури для них не створюються нові місця в пам'яті і вони використовують ті ж самі області пам'яті, які відведені відповідним фактичним параметрам. Тому усі зміни, які відбуваються над формальними параметрами, насправді відбуваються над їх фактичними образами. Для того, щоб параметри-змінні відрізняти від параметрів-значень, перед такими ідентифікаторами вказується службове слово **var**.

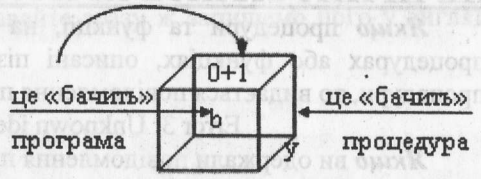
```
.....
procedure proba(x: integer; var y: integer);
  begin
     x:=x+1; y:=y+1
  end;
.....
  begin
.....
     a:=0; b:=0;
```



```
     proba(a,b);
     writeln(a,b);
.....
```

Дана програма надрукує такі результати: 0 1.

Зобразимо схематично роботу процедури з параметрами-змінними (мал.28).



Мал.28

Зробимо ще один висновок.

При зверненні до процедури параметрам-змінним не можна передавати значення сталих величин.

Це тому, що їх значення не можна змінювати!

Можна запропонувати такий вихід з положення: зробіть присвоювання значення сталої деякій змінній, а потім використайте цю змінну при зверненні до процедури.

Наприклад,

```
b:=0;
proba(0,b);
```

Питання для самоконтролю:

1 *Яка відмінність між глобальними та локальними змінними?*
2 *Які параметри називаються фактичними, а які формальними?*
3 *Що відбувається із змінними в пам'яті комп'ютера підчас виконання процедур і функцій та їх завершення?*
4 *Яким чином співвідноситься кількість та тип формальних і фактичних параметрів?*
5 *Що називається підпрограмою?*
6 *Які переваги при складанні програм надає використання підпрограм?*
7 *Які типи підпрограм ви можете назвати?*
8 *Які типи змінних можуть використовуватися в якості параметрів процедур та функцій?*
9 *Запишіть загальний вигляд функції.*
10 *Чи можуть існувати функції, в заголовку яких не вказані формальні параметри?*
11 *Чим відрізняються процедури від функцій? Запишіть загальний вигляд процедури.*
12 *Яка різниця між параметрами-значеннями та параметрами-змінними?*
13 *Поясніть, як відбувається розподіл пам'яті комп'ютера для параметрів-значень та параметрів-змінних.*

14 Чи можна формальному параметру-змінній ставити у відповідність сталу величину? Поясніть свою відповідь.

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Якщо процедури та функції, на які є посилання в інших процедурах або функціях, описані пізніше, ніж ці функції або процедури, то видається повідомлення про помилку

Error 3: Unknown identifier .

Якщо ви одержали повідомлення про помилку

Error 87: «,» expected

при зверненні до процедури або функції, то це швидше за все означає, що кількість фактичних параметрів не збігається з кількістю формальних.

Якщо при зверненні до процедури формальному параметру-змінній ви поставили у відповідність фактичний параметр-константу, то буде видане повідомлення про помилку

Error 20: Variable identefier expected .

Щоб позбутися цієї помилки, необхідно попередньо значення константи присвоїти якійсь змінній.

Якщо в основній програмі ви не одержуєте результатів, обчислених в процедурі, це означає, що формальні параметри, значення яких повертаються з процедури, не описані як параметри-змінні.

Значення функції в основній програмі повинно бути присвоєне деякій змінній того ж типу. Якщо ж ви цього не зробили, про це нагадає повідомлення про помилку

Error 122: Invalid variable reference .

Якщо в якості формальних параметрів ви випадково використали структуровані типи (масиви, обмежені рядкові величини), то компілятор видасть повідомлення про помилку

Error 12: Type identifier expected

або

Error 89: «)» expected .

42. Складання алгоритмів з використанням функцій

Приклад 1. Розглянемо алгоритм обчислення значення c=f(a,b)+f(a2,b2), якщо відомо, що f(x,y)=sinx+cosy. Звичайно, що можна підійти до розробки алгоритму обчислення значення с з математичної точки зору. Але давайте скористаємося можливостями алгоритмізації і розглянемо цю задачу саме з цієї точки зору.

Нескладно побачити, що можна окремо записати арифметичний вираз для обчислення значення функції в залежності від різних параметрів, а потім лише скористатися цим алгоритмом для обчислення результуючого значення змінної с. Алгоритм задачі настільки прозорий, що давайте зразу ж запишемо його у вигляді програми:

```
program value;
 var a,b,c: real;
function f(x,y: real): real;
  begin
    f:=sin(x)+cos(y);
  end;
  begin
    writeln ('Задайте значення двох параметрів');
    readln(a,b);
    c:=f(a,b)+f(sqr(a),sqr(b));
    writeln('Значення функції:', f:0:3)
  end.
```

Отож, основний алгоритм є лінійним. За рахунок створення допоміжного алгоритму ми можемо, звернувшись до нього з певними значеннями параметрів, отримати результат обчислення допоміжної функції і використати його при обчисленні значення результуючої величини.

Приклад 2. При зверненні до функцій у якості параметрів можна використовувати арифметичні вирази. А, як ми знаємо, арифметичні вирази можуть містити виклики функцій, тобто доходимо висновку, що параметром функції може бути виклик іншої функції. Нехай у нас є опис функції, що визначає більшу з двох заданих числових величин. Запишемо алгоритм-програму, яка визначає більшу з трьох числових величин, скориставшись описаною функцією.

```
program max_max;
 var a,b,c,M: integer;
function max(x,y: integer): integer;
  begin
    if x>y then max:=x else max:=y;
  end;
```

```
begin
   writeln ('Задайте три числа');
   readln(a,b,c);
   M:=max(max(a,b),c);
   writeln('Максимальне з трьох заданих чисел:', M)
end.
```

Ось і все: ми спочатку звернулись до функції max з параметрами *a*, *b*, а потім з результатом цього звернення та із значенням змінної *c* знову звернулися до цієї ж функції, не порушивши кількості фактичних параметрів. Основний алгоритм-програма у нас вийшов лінійним.

Приклад 3. Розглянемо приклад створення функції. Нехай нам необхідно обчислити таке значення: z=x!+y!

Оскільки факторіал – це одне число, то нас влаштує функція, яка буде обчислювати його значення. Оформимо відомий нам алгоритм у вигляді підпрограми-функції.

```
function factorial(n: integer): longint;
 var i: integer; f: longint;
begin
   f:=1;
   for i:=1 to n do
       f:=f*i;
   factorial:=f;
end;
```

Використати цю функцію в основному алгоритмі-програмі для обчислення значення z ми зможемо таким чином:

```
writeln ('Задайте x та y:');
readln(x,y);
z:= factorial(x) + factorial(y);
writeln('Сума факторіалів заданих чисел: ',z);
```

Зверніть увагу на те, що оскільки після виконання функції її ім'я «тримає в собі» обчислене значення, то в основній програмі його треба кудись обов'язково «покласти». Це означає, що звернення до функції повинно використовуватись лише в операторах присвоювання або в процедурі виведення інформації **write** (**writeln**).

Питання для самоконтролю:

1 *Які правила опису і використання функцій?*
2 *Як здійснюється передача результату обчислення функції в основну програму?*
3 *Як здійснюється виклик функції?*
4 *Чи можна в якості параметра функції знову використовувати виклик функції?*
5 *Наведіть свій приклад функції і її виклику в основній програмі та реалізуйте його.*

43. Складання алгоритмів з використанням процедур

Приклад 1. Перепишемо функцію знаходження значення факторіала у вигляді процедури і після цього порівняємо отримані алгоритми-програми.

```
program max_max;
  var x,y: integer; f_x,f_y,f_sum: longint;
  procedure factorial(n: integer; var f: longint);
    var i : integer;
    begin
        f:=1;
        for i:=1 to n do
            f:=f*i;
    end;
  begin
    write ('Задайте два числа: ');
    readln(x,y);
    factorial(x,f_x);
    factorial(y,f_y);
    f_sum:=f_x+f_y;
    writeln('Сума значень факторіалів заданих чисел:', f_sum)
  end.
```

Якщо обчислене значення факторіалу числа при використанні функції передавалося в основну програму через ім'я цієї функції, то в цьому прикладі доводиться вводити параметр-змінну **var** f: **integer**, яка міститиме обчислене значення в процедурі і передасть його в основну програму.

Звичайно, що в нашому випадку має перевагу запис даного алгоритму з використанням функції. Це в першу чергу звернення до допоміжного алгоритму та отримання результату.

Зауважте, що виклик процедури є лінійним оператором, оскільки після його виконання ми завжди переходимо до виконання наступного записаного в програмі оператора.

Якщо в попередньому прикладі для створення підпрограми можна було використати як функцію, так і процедуру, то існують алгоритми, де без організації процедури не обійтися.

Приклад 2. Запишемо невеликий корисний алгоритм у вигляді програми, використавши в ньому процедуру, яка перетворює всі маленькі літери заданого тексту на великі і обчислює, скільки таких перетворень було зроблено.

```
program invers;
  uses crt;
  var txt:      string;
      count:    integer;
procedure letters(var S: string, var n: integer);
  var i: integer;
  begin
     n:=0;
     for i:=1 to Length (S) do
        if S[i]<> UpCase(S[i]) then
           begin
              S[i]:=UpCase (S[i]);
              Inc(n)
           end
  end;
  begin
     write ('Введіть текст: ');
     readln (txt);
     letters(txt,count);
     writeln ('Перетворений текст:',txt,);
     writeln ('В заданому тексті було ',count,' маленьких літер');
     repeat until KeyPressed
  end.
```

В цій програмі використана нова функція Паскаля **UpCase**. Вона перетворює маленькі літери у відповідні їм великі. Формальні параметри S та n в процедурі є параметрами-змінними, тобто містять обчислені в процедурі результати і передають їх в основну програму.

Спробуйте переписати функцію, що робить «гортання сторінок» вашої програми, у вигляді процедури. Це буде раціональніше.

Приклад 3. При поясненні понять параметрів-значень та параметрів-змінних ми зазначили, що вони можуть бути описані лише простими типами. Виходить, що виникає проблема при передаванні у функцію масивів в якості параметрів. Можна здогадатись, що якимось чином ця проблема в Паскалі все ж таки вирішується. Для того, щоб із структурованого типу «масив» утворити простий тип, необхідно в розділі type описати свій власний тип, надавши йому унікальне ім'я в даній програмі. Нехай цей опис буде виглядати таким чином:

```
type mas=array[1..100] of real;
```

Тепер у програмі ми можемо використовувати змінні типу mas, які в розділі змінних var будуть виглядати як прості:

```
var a: mas;
```

Введення поняття власних типів користувача – це потужній засіб мови програмування Паскаль, але в межах даного посібника ми його розглядати не будемо. Скористаємося розділом type лише для передавання масивів в якості параметрів у підпрограми.

Програма, яку ми розглянемо, буде стосуватися реалізації алгоритму складання і використання процедури і функції, що відповідно вводять значення двох одновимірних масивів та визначають їх максимальні значення. Задавши в якості вхідних даних два масиви, визначимо суму їх максимальних значень.

```
program sum_max;
  uses crt;
  type mas=array[1..100] of real;
  var a,b:      mas;
      n,m:      integer;
      sum:      real;
procedure input_mas(var x: mas; k: integer);
    var i:      integer;
    begin
       for i:=1 to k do
       readln(x[i]);
end;
```

```
function max_mas(x: mas; k: integer): real;
   var i:      integer;
       max:    real;
   begin
     max:=x[1];
     for i:=2 to k do
       if x[i]<max then max:=x[i];
     max_mas:=max
   end;
begin
   write ('Задайте кількість елементів першого масиву: ');
   readln(n);
   writeln('Задайте значення елементів першого масиву:');
   input_mas(a,n);
   write ('Задайте кількість елементів другого масиву: ');
   readln(m);
   writeln('Задайте значення елементів другого масиву:');
   input_mas(b,m);
   sum:=max_mas(a,n)+max_mas(b,m);
   writeln('Сума максимальних значень двох масивів:',sum:0:3);
   repeat until KeyPressed
end.
```

Заголовок процедури введення значень елементів масивів **procedure** input_mas(**var** x: mas; k: **integer**) підказує нам, що кількість елементів масиву, що задається, передається в процедуру як параметр-значення, а сам масив, тобто його значення, повертається в основну програму, як параметр-змінна, що описана власним типом користувача. Заголовок функції обчислення максимального значення масиву **function** max_mas(x: mas; k: **integer**): **real** має два формальні параметри-значення, оскільки результат обчислення – одне число – передається в основну програму через ім'я функції. В решті-решт основна програма набула лінійної структури.

Приклад 4. Розглянемо ще один приклад процедури, яка стискає заданий текст, вилучаючи з нього всі пробіли.

```
procedure compress(var S: string);
  var i: integer;
begin
   i:=Pos(' ',S);
   while i<>0 do
     begin
       Delete(S,i,1);
       i:=Pos(' ',S)
     end
end;
```


Корисна порада. Бувають ситуації, коли перша процедура або функція використовує в собі звернення до другої, а друга – звернення до першої. Оскільки в Паскалі розподіл пам'яті та виконання програми йде в тій послідовності, в якій вони записані, то виникає питання: а яку процедуру чи функцію описати першою? Для цього існує можливість випереджаючого опису. Необхідно заголовок другої за чергою процедури або функції винести перед першою із службовим словом forward, а саму процедуру описати після першої. Наведемо приклад:

```
procedure a(<список формальних параметрів>);forward;
procedure b(<список формальних параметрів>);
.....
begin
.....
   a(<список фактичних параметрів>);
end;
procedure a;
.....
begin
.....
   b(<список фактичних параметрів>);
end;
.....
```

Ми завершуємо розділ процедур та функцій. Чого ми навчилися в ньому, якою корисною інформацією можемо скористатися надалі? А навчилися ми елементам структурного програмування, вмінням виділяти основні моменти програми в окремі блоки, які називаються процедурами та функціями. Такий процес, коли ми спочатку аналізуємо всю задачу, виділяючи в ній глобальні проблеми (основний алгоритм), і лише потім деталізуємо їх

(допоміжні алгоритми), при необхідності виділяючи в них нові проблеми (допоміжні алгоритми нижчого рівня), називається програмуванням «зверху – вниз».

Питання для самоконтролю:

1 *В чому полягає специфіка використання процедур?*
2 *Які переваги при складанні програм надає використання процедур?*
3 *Чи завжди потрібно надавати перевагу процедурам?*
4 *Яким чином у Паскалі реалізована можливість передавання значень елементів масиву у підпрограму?*
5 *Наведіть власний приклад алгоритму з використанням процедури і реалізуйте його у вигляді програми.*
6 *Що ми розуміємо під програмуванням «зверху – вниз»?*

44. Практична робота «Програми із зверненнями до підпрограм»[9]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм із зверненням до підпрограм.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму або його фрагментів, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму.*
6) *Визначити типи підпрограм, які необхідно застосувати для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання функцій та процедур, різних базових алгоритмічних структур для реалізації алгоритму.*
9) *Записати алгоритм мовою програмування.*

[9] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Звернення до підпрограм»

10) *Набрати текст програми, використовуючи середовище програмування та виконати налагодження програми, виправивши синтаксичні помилки.*
11) *Виконати програму, підготувавши систему тестів.*
12) *Реалізувати різні варіанти використання функцій і процедур для виконання сформульованої задачі та проаналізувати їх щодо кількості виконуваних дій.*

Створення графічних зображень

45. Процедури і функції побудови графічних зображень

Якщо ви зразу розгорнули книжку на цій сторінці, вас можна зрозуміти, але й одночасно слід застерегти. Цей розділ навмисне винесений на самий кінець. А пов'язане це з тим, що без гарних алгоритмічних навиків не можна на повну потужність скористатися графічними можливостями будь-якої мови програмування. Отже, почніть знайомство з посібником спочатку і скористайтеся усіма корисними порадами, які містяться в ньому.

На жаль, розміри цього посібника не дозволяють приділити графіці стільки уваги, скільки вона заслуговує. Відомі програмісти присвячують їй товстелезні книжки, щедро діляться в них дорогоцінною інформацією та своїм власним досвідом.

Всі процедури та функції для роботи в графічному режимі містяться у модулі **Graph**. Тому його необхідно підключати на початку роботи з програмою.

Перш, ніж почати роботу в графічному режимі, необхідно переключити програму у режим читання та запису інформації на екран по пікселях. Для цього необхідно виконати ***процедуру ініціалізації графіки***:

InitGraph(GraphDriver, GraphMode, GraphPath),

де GraphDriver – ціле число, що визначає тип монітора, GraphMode – ціле число, що визначає режим роботи цього монітора, GraphPath – рядковий вираз, який визначає шлях до файла-драйвера egavga.bgi.

Розглянемо детальніше призначення параметрів, які використовуються при ініціалізації графіки.

Параметр GraphDriver задається цілим числом або мнемонічним позначенням і означатиме, який тип графічного адаптера встановлений на комп'ютері чи з яким ви хочете працювати. Наприклад, якщо встановлений монітор типу VGA, але ви хочете працювати в режимі CGA – це можливо. Значення параметра GraphDriver можна взяти з таблиці 6:

Таблиця 6

Ім'я	Значення
Detect	0 (Автоматичний вибір драйвера)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMono	5
IBM8514	6
HercMono	7
ATT400	8
VGA	9
PC3270	10
CurrentDriver	-128 (Поточний драйвер)

Параметр GraphMode визначає, в якому режимі роздільної здатності буде працювати обраний вами графічний адаптер. В наступній таблиці 7 наведені лише можливі режими найпоширеніших типів адаптерів:

Таблиця 7

Ім'я	Значення	Розмір поля	Палітра
CGAC0	0	320*200	C0
CGAC1	1	320*200	C1
CGAC2	2	320*200	C2
CGAC3	3	320*200	C3
CGAHi	4	640*200	2 кольори
EGALo	0	640*200	16 кольорів
EGAHi	1	640*350	16 кольорів
EGA64Lo	0	640*200	16 кольорів
EGA64Hi	1	640*350	4 кольори
EGAMonoHi	0	640*350	2 кольори

Ім'я	Значення	Розмір поля	Палітра
VGALo	0	640*200	16 кольорів
VGAMed	1	640*350	16 кольорів
VGAHi	2	640*480	16 кольорів

C0 – світло-зелений, рожевий, жовтий; C1 – світло-блакитний, світло-фіолетовий, білий; C2 – зелений, червоний, коричневий; C3 – блакитний, фіолетовий, світло-сірий.

Параметр GraphPath – це рядкова величина, яка визначає, де знаходиться драйвер графічного введення-виведення інформації. Для різних типів адаптерів існують різні драйвери: для моніторів EGA та VGA – egavga.bgi, для монітора CGA – cga.bgi. Якщо цей файл міститься в тому ж самому підкаталозі, що й exe-файл програми, то шлях не вказується (''), якщо ж в іншому місці, то його необхідно вказати (наприклад, 'c:\BGI').

Надалі ми будемо використовувати такий вигляд процедури ініціалізації графіки:

```
var GraphDriver, GraphMode, GraphPath: integer;
.....
GraphDriver:=VGA;
GraphMode:=VGAHi;
InitGraph(GraphDriver, GraphMode, GraphPath);
.....
```

Процедура закриття графіки

CloseGraph

дозволяє програмі повернутися назад у текстовий режим. Бажано кожну програму, в якій передбачається робота з графікою, завершувати цією процедурою.

Далі розглянемо основні процедури та функції модуля **Graph**.

1. Процедура очищення екрана монітора в графічному режимі:

ClearDevice.

2. Процедура, що виводить зображення пікселя заданого кольору:

PutPixel (x,y: **integer**; Color: **Word**) ,

де x, y – координати точки, Color – колір точки. Значення кольорів залишаються ті самі, що й в текстовому режимі.

3. Процедура виведення лінії:

Line (X1, Y1, X2, Y2: **integer**) ,

де X1, Y1– координати початку лінії, а X2, Y2 – координати її кінця.

Для задання кольору лінії необхідно скористатися процедурою SetColor, яка буде описана пізніше.

4. Для виведення зображення кола треба скористатися процедурою

Circle(x,y: **integer**; Radius: **word**),

де (x,y) – координати центра кола, Radius – розмір радіуса цього кола. Колір кола задається таким самим чином, як і колір лінії.

5. Процедура

Ellipse (x,y: **integer**; StartAngle, EndAngle , XRadius,YRadius: **word**)

дозволяє вивести на екран монітора еліпс з центром у точці (x,y), радіусами по горизонталі та вертикалі відповідно Xradius, YRadius від кута StartAngle до кута EndAngle.

6. Побудова прямокутника відбувається за допомогою процедури

Rectangle (x1, y1, x2, y2: **integer**),

де (x1, y1) – координати лівого верхнього кута прямокутника, (x2, y2) – координати правого нижнього його кута.

7. Всі перераховані вище графічні примітиви створюються за допомогою ліній. Для задання їм певного кольору перед використанням відповідної процедури необхідно скористатися такою процедурою

SetColor (color: **word**),

де параметр color визначає необхідний колір ліній фігури.

8. Колір фону малюнка можна задати за допомогою процедури

SetBkColor (color: **word**).

9. Процедура, що пропонується, буде необхідна для використання у парі з наступними процедурами. На відміну від попередньої, вона встановлює колір фігури, що є частиною площини. Наприклад, прямокутник не як контур (**Rectangle**), а як частина площини (**Bar**).

Ось ця процедура:

SetFillStyle (Pattern: **word**; Color: **word**),

де Pattern визначає вид шаблону заповнення, а Color – його колір. Модуль **Graph** пропонує великий вибір значень параметра Pattern, які поміщені в таблицю 8:

Таблиця 8

Ім'я	Значення	Дія
EmptyFill	0	суцільне заповнення кольором фону
SolidFill	1	суцільне заповнення поточним кольором
LineFill	2	заповнення типу ===
LtSlashFill	3	заповнення типу ///
SlashFill	4	заповнення жирними лініями типу ///
BkSlashFill	5	заповнення жирними лініями типу \\\
LtBkSlashFill	6	заповнення типу \\\
HatchFill	7	заповнення густою штриховкою
XHatchFill	8	заповнення рідкою штриховкою
InterleaveFill	9	заповнення переривною лінією
WideDotFill	10	заповнення рідкими крапками
CloseDotFill	11	заповнення густими крапками
UserFill	12	власне заповнення

10. Зображення зафарбованого прямокутника виконується за допомогою процедури

Bar (x1, y1, x2, y2: **integer**).

11. Для малювання стовпчастих діаграм (паралелепіпедів) зручно використовувати процедуру

Bar3D (x1, y1, x2, y2: **integer**; D3: **word**; Top: **boolean**),

де глибина задається параметром D3, а параметр Top задає режим зображення верхньої площини (True – відображати, False – ні).

12. Для малювання кругових діаграм необхідно виділити деякий сектор заданого круга. Для цього можна скористатися процедурою

PieSlice (x, y: **integer**; StAngle, EndAngl, Radius: **word**),

де (x,y) – координати центра кола, Radius – розмір радіуса цього кола, а сам сектор задається значеннями двох кутів в градусній мірі: StAngle – початковий кут, EndAngl – кінцевий кут.

13. Для зображення сектора еліпса існує процедура

Sector (x, y: **integer**; StAngle, EndAngl, Xradius, YRadius: **word**),

де (x,y) – координати центра еліпса, XRadius – розмір радіуса еліпса по горизонталі, Yradius – розмір радіуса еліпса по вертикалі, StAngle – початковий кут сектора, EndAngl – кінцевий кут сектора.

14. І, кінець кінцем, універсальна процедура

FloodFill (x,y: **integer**; Border: **word**)

заповнює всю область навколо точки (x,y), що обмежена лініями кольору Border. Колір і тип заливки визначається процедурою **SetFillStyle**.

15. Функція **GraphResult** : **integer** повертає після ініціалізації графіки ціле число, яке є кодом завершення ініціалізації. Якщо цей результат 0, то він означає коректне завершення ініціалізації.

16. Функція **GetMaxX**: **integer** визначає максимальне значення пікселя по горизонталі у вибраному графічному режимі. При цьому початок відрахунку пікселів (0,0) знаходиться у верхньому лівому куті.

17. Функція **GetMaxY**: **integer** визначає максимальне значення пікселя по вертикалі у вибраному графічному режимі.

18. Функції **GetX**: **integer** та **GetY**: **integer** визначають поточне положення графічного курсора по горизонталі (X) та вертикалі (Y).

Далі ми будемо розглядати процедури, що стосуються виведення текстової інформації в графічному режимі. Оскільки після ініціалізації графіки будь-які зображення виводяться по пікселях, то це стосується і текстової інформації. Таке виведення тексту відбувається набагато повільніше, ніж у тескстовому режимі.

19. Для вибору певного шрифту існує процедура

SetTextStyle (Font, Direction: **word**; CharSize: **word**),

де Font задає один з типів шрифтів, Direction – горизонтальне (0) чи вертикальне (1) виведення тексту, а CharSize – розмір символів шрифта (від 1 до 10).

Паскаль дозволяє працювати з декількома стандартними шрифтами (табл.9):

Таблиця 9

Ім'я	Значення	Вигляд
DefaultFont	0	матричний шрифт 8*8 (по замовчуванню)
TriplexFont	1	напівжирний шрифт
SmallFont	2	світлий шрифт (тонке накреслення)
SansSerifFont	3	книжна гарнітура (рубаний шрифт)
GothicFont	4	готичний шрифт

Для роботи з шрифтами в графічному режимі необхідно, щоб в поточному каталозі були файли з розширенням *.chr, які і складають набір штрихових шрифтів. Перший шрифт зрозумілий

програмі без додаткової інформації, а інші реалізовані у відповідних файлах trip.chr, small.chr, sans.chr, goth.chr.

20. Перш ніж скористатися в програмі додатковими шрифтами, їх необхідно зареєструвати. Для цього існує функція

InstallUserFont (FontileName: **string**): **integer** ,

яка повертає наступний вільний номер, під яким буде зареєстрований даний шрифт.

21. Безпосереднє виведення тексту можна здійснити за допомогою процедури

OutTextXY (x,y: **integer**; TextString: **string**) ,

де (x,y) – координати лівого верхнього кута початку тексту, а TextString – безпосередньо сам текст або змінна, що його містить.

Питання для самоконтролю:

1 *В чому полягає особливість роботи користувача в графічному режимі?*
2 *Що розуміється під ініціалізацією графіки? Яким чином відбувається налагодження програми на роботу з необхідним типом монітора у певному режимі?*
3 *Які процедури модуля Graph дозволяють виводити на екран монітора прості геометричні фігури?*
4 *Які процедури модуля Graph дозволяють працювати з кольорами? Які їх особливості?*
5 *Які процедури та функції модуля Graph реалізують роботу з текстами?*
6 *В чому полягає особливість виведення тексту в графічному режимі роботи монітора?*
7 *Наведіть власний приклад програми для роботи з графікою.*

НЕ ПРИПУСКАЙТЕСЯ ПОМИЛОК!

Якщо при зверненні до процедур або функцій модуля **Graph** ви одержуєте повідомлення про помилку

Error 3: Unknown identifier ,

то це означає, що ви забули підключити модуль **Graph**.

Якщо компілятор на назву модуля **Graph** в розділі **Uses** реагує виведенням повідомлення про помилку

Error 15: File not found ,

то це може означати, що в поточному підкаталозі відсутній модуль **Graph**.

Якщо в поточному підкаталозі відсутній файл egavga.bgi, то ви одержите повідомлення про помилку на кроці виконання програми

BGI Error: Graphics not initialized (use InitGraph) .

Якщо в програмі ви виконуєте виведення графічних образів, а на кроці її виконання не бачите їх на екрані, то це може відповідати таким помилковим ситуаціям:

- колір графічного образу співпадає з кольором фону;
- графічні образи виводяться поза межами графічного екрана.

46. Складання алгоритмів створення графічних зображень

Приклад 1. Спробуємо підібрати приклад програми, що вмістила більшість означених вище графічних можливостей. При виконанні цієї програми ви побачите на екрані монітора покрокове виведення кіл різного кольору, зафарбовування їх різними кольорами, виведення зображення елемента стовпчастої діаграми, виведення тексту в графічному режимі.

```
program picture;
  uses Graph;
  var GraphDriver, GraphMode: integer;
begin
  GraphDriver:=VGA;
  GraphMode:=VGAHi;
  InitGraph (GraphDriver, GraphMode,'');
  SetTextStyle(1,0,4);
  OutTextXY(50,400,'Перші графічні етюди.');
  SetColor (red);
  Circle(50,50,20);
  readln;
  SetColor(green);
  Circle(50,50,40);
  readln;
  SetFillStyle(1, blue);
  FloodFill (50,50, red);
  readln;
  SetFillStyle(11, magenta);
  FloodFill (50,50, green);
  readln;
  ClearDevice;
  SetFillStyle (4, yellow);
  Bar3D (10,10,50,60,10, True);
  readln;
```



```
  CloseGraph;
end.
```

Для перевірки отриманої інформації зробіть зміни в цій програмі і виконайте її. Наприклад, поміняйте розміщення і розміри кіл, кольори і тип їх зафарбовування, шрифт тексту, сам текст тощо.

Приклад 2. Використання комп'ютера для побудови і дослідження графіків різних функцій – дуже корисна річ в розділі процесів моделювання. Спершу це здається дуже просто: достатньо організувати цикл із зміною значень аргумента, обраховувати значення відповідної функції і виводити на екран монітора точки з цими координатами. Але при цьому ми не враховуємо те, що координати точок на екрані монітора мають лише цілі невід'ємні значення, початок координат знаходиться в лівому верхньому куті, а зростання по вісі OY відбувається вниз. Щоб зображення графіка було максимально наближеним до реального, необхідно оцінити можливі зміни значень як аргументу, так і результату функції, що будується, і ввести масштабування. Розглянемо приклад побудови графіка функції $\left|\sin x\right|$.

```
program graphic;
  uses Graph;
  var i,GraphDriver, GraphMode: integer;
begin
  GraphDriver:=VGA;
  GraphMode:=VGAHi;
  InitGraph (GraphDriver, GraphMode,'');
  for i:=1 to 600 do
    PutPixel(i,200-round(abs(sin(i/30)*50)),4);
  readln;
  CloseGraph;
end.
```

Для того, щоб зручніше було розібратися з цією програмою, прокоментуємо деякі коефіцієнти у виразі 200-**round**(**abs**(**sin**(i/30)*50)), який у загальному вигляді можна записати таким чином: K-**round**(**abs**(**sin**(i/L)*M)). Для отримання бажаної щільності точок, якими зображується графік функції, необхідно ввести відпо-

відний коефіцієнт L. Амплітуда синусоїди залежить від коефіцієнта M, а величина K дає змогу зсунути графік вниз по вісі OY.

Замінивши функцію на якусь іншу, можна отримати і дослідити її графік, змінюючи відповідні параметри.

Наприклад, такі фрагменти програм:

```
for i:=1 to 200 do
   PutPixel(i,400-round(exp(i/20)*7),2);
```

та

```
for i:=1 to 600 do
   PutPixel(i,200-round(ln(i*10)*7),2);
```

дозволять поекспериментувати з графіками функцій exp(x) та ln(x).

Приклад 3. Дуже часто доводиться виводити на екран монітора геометричні фігури. Але при цьому слід враховувати специфіку екранних координат: точка з координатами (0,0) виводиться на екран монітора у лівому верхньому куті, збільшення по вісі OX іде зліва направо, а от по вісі OY – зверху вниз. Отже, бачимо, що співпадань з геометричною побудовою у декартовій системі координат не так вже і багато. Тому виникає необхідність у враховуванні масштабування в разі великих значень параметрів точок на координатній площині, що виходять за межі екранних координат, відсутності від'ємних та дробових значень при виведенні точок на екран монітора. В якості прикладу програми, що масштабує побудову многокутника, заданого координатами своїх вершин в порядку їх обходу, при виведенні на екран монітора, можна навести таку.

```
program picture_1;
 uses Graph;
 var
    i,GraphDriver, GraphMode, n: integer;
    a,b:                         array[1..100] of real;
    point_a,point_b:             array[1..100] of integer;
    mina, maxa, minb, maxb:      real;
    kx, ky, k:                   real;
 begin
    writeln('input n:');
    read(n);
    writeln('input data:');
    for i:=1 to n do
       read(a[i],b[i]);
```



```
    {визначення максимальних значень по вісі OX та по вісі OY}
    maxa:=abs(a[1]);        maxb:= abs(b[1]);
    for i:=2 to n do
      begin
        if abs(a[i])>maxa then maxa:= abs(a[i]);
        if abs(b[i])>maxb then maxb:= abs(b[i]);
      end;
    kx:=1; ky:=1;
    {визначення коефіцієнта масштабування}
    if maxa>319 then kx:=319/maxa;
    if maxb>239 then ky:=239/maxb;
    if kx>ky then k:=ky else k:=kx;
    {перерахунок значень координат точок}
    {з урахуванням коефіцієнта масштабування}
    for i:=1 to n do
      begin
        point_a[i]:=round(a[i]*k);
        point_b[i]:=round(b[i]*k);
      end;
    {переміщення центру координат у центр екрана монітора}
    for i:=1 to n do
      begin
        point_a[i]:=point_a[i]+320;
        point_b[i]:=240-point_b[i];
      end;
    {виведення зображення многокутника}
    {на екран монітора у графічному режимі}
    GraphDriver:=VGA;
    GraphMode:=VGAHi;
    InitGraph (GraphDriver, GraphMode,'');
    Line(320,0,320,479);
    Line(0,240,639,240);
    SetColor(2);
    for i:=1 to n-1 do
    Line(point_a[i],point_b[i],point_a[i+1],point_b[i+1]);
    Line(point_a[n],point_b[n],point_a[1],point_b[1]);
    readln;
    CloseGraph;
end.
```

Питання для самоконтролю:

1 *Як організувати покрокове виведення графічних зображень на екран монітора?*
2 *В чому полягає специфіка виведення графіків функцій на екран монітора?*
3 *Для чого використовується масштабування при виведенні графічних зображень на екран монітора?*
4 *Як визначити масштаб для виведення графічних зображень на екран?*
5 *Реалізуйте власний приклад програми для роботи з графікою.*

47. Практична робота «Побудова графічних зображень»[10]

За наведеним сценарієм виконайте завдання по створенню та налагодженню алгоритмів та програм з використанням графічних зображень.

1) *Визначити кількість і типи вхідних даних відповідно до умови задачі.*
2) *Визначити кількість і типи вихідних даних відповідно до умови задачі.*
3) *Визначити тип алгоритму або його фрагментів, побудувати математичну модель.*
4) *Визначити необхідність введення проміжних даних, їх кількість та типи.*
5) *Розробити словесний опис алгоритму.*
6) *Визначити графічні операції, які необхідно застосувати для створення алгоритму відповідно до умови задачі.*
7) *Визначити послідовність введення початкових та виведення результуючих даних з використанням відповідних коментарів і форматування.*
8) *Визначити коректність використання різних базових алгоритмічних структур для побудови графічних зображень.*
9) *Записати алгоритм мовою програмування.*

[10] Завдання для даної практичної роботи наведені у «Збірнику вправ та задач з алгоритмізації та програмування» у розділі «Побудова графічних зображень»

10) *Набрати текст програми, використовуючи середовище програмування.*
11) *Виконати налагодження програми, виправивши синтаксичні помилки.*
12) *Виконати програму, підготувавши систему тестів.*

48. Тематична атестація «Процедури і функції. Графічні операції»

Для виконання тематичної атестації необхідно:

знати

- поняття основного та допоміжного алгоритму;
- поняття глобальних та локальних змінних;
- означення фактичних та формальних параметрів;
- типи підпрограм;
- опис функцій та процедур мовою програмування;
- переваги застосування функцій чи процедур в конкретних алгоритмах;
- особливості роботи в текстовому та графічному режимах;
- функції та процедури для роботи в графічному режимі монітора;
- особливості роботи з текстовою інформацією в графічному режимі монітора;
- особливості використання масштабування при виведенні графічних зображень;

вміти

- описувати мовою програмування функції та процедури;
- правильно організовувати виклик функцій та процедур;
- аналізувати коректне використання функцій та процедур в конкретних алгоритмах;
- наводити власні приклади опису функцій та процедур;
- розв'язувати задачі з використанням функцій та процедур;
- ініціалізувати графічний режим роботи монітора;
- коректно використовувати функції та процедури графічного модуля;
- коректно використовувати масштабування при виведенні графічних зображень на екран монітора;
- розв'язувати задачі з використанням графічних зображень.

«Гарячі клавіші» середовища Turbo Pascal

«Гаряча клавіша»	*Дія*
F1	Одержання довідки (допомоги)
Ctrl+F1	Активізація довідки про оператор, на який указує курсор
F2	Збереження у файлі тексту з активного вікна
Ctrl+F2	Закриття всіх відкритих програмою файлів і встановлення програмного лічильника на початок програми
F3	Відкриття нового вікна і завантаження у нього вказаного файла
Alt+F3	Закриття активного вікна
F4	Запуск на виконання програми до позиції курсора
Ctrl+F4	Аналіз і зміна значень змінних
F5	Збільшення/зменшення розміру активного вікна
Alt+F5	Перемикання на екран користувача
Ctrl+F5	Зміна положення і розмірів активного вікна
F6	Перехід до наступного вікна
Shift+F6	Перехід до попереднього вікна
Alt+цифра	Перехід до вікна з вказаним номером
F7	Виконання програми пооператорно з пооператорним виконанням усіх підпрограм
F8	Виконання програми пооператорно з виконанням усіх підпрограм без операторної деталізації
Ctrl+F7	Доповнення списку змінних у **Watch**-вікні
Ctrl+F8	Встановлення/скасування контрольної точки на рядку програми, що вказана курсором
Alt+F9	Компіляція програми з активного вікна
Ctrl+F9	Компіляція і запуск програми на виконання з активного вікна
F10	Активізація головного меню
Ctrl+Y	Вилучення рядка, на який вказує курсор
Ctrl+N	Вставлення нового рядка після того, на який указує курсор
Shift+стрілки	Розширення блока, що відмічається, від положення курсора в напрямку стрілки

Shift+END	Розширення блока, що відмічається, від положення курсора до кінця рядка
Shift+HOME	Розширення блока, що відмічається, від положення курсора до початку рядка
[Ctrl+K]+[B]	Вказання початку блока, що відмічається
[Ctrl+K]+[K]	Вказання кінця блока, що відмічається
[Ctrl+K]+[H]	Зняття/відновлення відміченого блока
[Ctrl+K]+[C]	Копіювання відміченого блока в те місце, де встановлено курсор
[Ctrl+K]+[V]	Перенесення відміченого блока в те місце, де встановлено курсор
Ctrl+Del	Знищення відміченого блока
Ctrl+Ins	Копіювання відміченого блоку в буфер проміжного зберігання
Shift+Del	Перенесення відміченого блоку в буфер проміжного зберігання
Shift+Ins	Копіювання відміченого блока з буфера проміжного зберігання в те місце, де встановлено курсор
Alt+X	Завершення сеансу роботи інтегрованого середовища Turbo Pascal із збереженням (в разі підтвердження) змінених файлів

Основні помилки компіляції середовища Turbo Pascal 7.0

1 Out of memory – вихід за межі пам'яті
2 Identifier expected – очікується ідентифікатор
3 Unknown identifier – невідомий ідентифікатор
4 Duplicate identifier – повторний опис ідентифікатора
5 Syntax error – синтаксична помилка
6 Error in real constant – помилка в дійсній константі
7 Error in integer constant – помилка в цілій константі
8 String constant exseeds line – рядкова константа перевищує розміри рядка
9 Too many nested files – забагато вкладених файлів
10 Unexpected end of file – несподіваний кінець файла
11 Line too long – рядок надто довгий
12 Type identifier expected – очікується ідентифікатор типу

13 Too many open files – надто багато відкритих файлів
14 Invalid file name – помилкове ім'я файла
15 File not found – файл не знайдено
16 Disk full – диск заповнений
17 Invalid compiler directive – невірна директива або ключ компілятора
18 Too many files – надто багато файлів
19 Undefined type in pointer definition – невизначений тип у визначенні посилання
20 Variable identifier expected – потрібен ідентифікатор змінної
21 Error in type – помилка у визначенні типу
22 Structure too lange – надто велика структура
23 Set base type out of range – базовий тип множини порушує дозволені межі
24 File components may not be files or objects – компоненти файла не можуть бути файлами або об'єктами
25 Invalid string length – неправильна довжина рядка
26 Type mismatch – невідповідність типів
27 Invalid subrange base type – невірний базовий тип для діапазону
28 Lower bound > then upper bound – нижня межа більша за верхню
29 Ordinal type expected – потрібен зчислений тип
30 Integer constant expected – очікується ціла константа
31 Constant expected – очікується константа
32 Integer or real constant expected – очікується ціла або дійсна константа
33 Pointer type identifier expected – очікується ім'я типу покажчик
34 Invalid function result type – неправильний тип результату функції
35 Label identifier expected – потрібен ідентифікатор мітки
36 BEGIN expected – очікується **BEGIN**
37 END expected – очікується **END**
38 Integer expression expected – очікується вираз цілого типу
39 Ordinal expression expected – очікується вираз зчисленного типу
40 Boolean expression expected – очікується логічний вираз
41 Operand types do not match operator – типи операндів не відповідають оператору
42 Error in expression – помилка у виразі
43 Illegal assignment – хибне присвоєння

44 Field identifier expected – очікується ім'я поля запису
45 Object file too large – об'єктний файл занадто великий
46 Undefined EXTERN – невизначена зовнішня процедура
47 Invalid object file record – неправильний запис об'єктного файла
48 Code segment too large – сегмент коду надто великий
49 Data segment too large – сегмент даних надто великий
50 DO expected – очікується слово **DO**
51 Invalid PUBLIC definition – неправильне визначення **PUBLIC**
52 Invalid EXTRN definition – неправильне визначення **EXTRN**
53 Too many EXTRN definition – надто багато визначень типу **EXTRN**
54 OF expected – очікується слово **OF**
55 INTERFACE expected – очікується інтерфейсна секція
56 Invalid relocatable reference – неприпустиме переміщуване посилання
57 THEN expected – очікується слово **THEN**
58 TO or DOWNTO expected – очікується слово **TO** або **DOWNTO**
59 Undefined forward – невизначений випереджаючий опис
60 Too many procedures – дуже багато процедур
61 Invalid typecast – неправильне приведення типу
62 Division by zero – ділення на нуль
63 Invalid file type – неправильне файловий тип
64 Cannot Read or Write variables of this type – немає можливості читати або записати змінні даного типу
65 Pointer variable expected – очікується змінна-показчик
66 String variable expected – очікується рядкова змінна
67 String expression expected – очікується вираз рядкового типу
68 Unit not found – програмний модуль не знайдено
69 Unit name mismatch – невідповідність імен програмних модулів
70 Unit version mismatch – невідповідність версій програмних модулів
71 Internal stack overflow – переповнення внутрішнього стека
72 Unit file format error – помилка формату файла програмного модуля
73 Implementation expected – очікується секція реалізації
74 Constant and case types don't match – типи константи та тип виразу оператора **case** не відповідають одне одному

75 Record or object variable expected – очікується змінна типу «запис»
76 Constant out of range – значення константи виходить за межі допустимих значень
77 File variable expected – очікується файлова змінна
78 Pointer expression expected – очікується вираз типу покажчика
79 Integer or real expression expected – очікується вираз цілого або дійсного типу
80 Label not within current block – мітка не знаходиться в середині поточного блоку
81 Label already defined – мітка вже визначена
82 Undefined label in proceding statement part – невизначена мітка у виконуваному розділі операторів
83 Invalid @ argument – неправильний аргумент оператора @
84 Unit expected – очікується слово **Unit**
85 «;» expected – очікується «;»
86 «:» expected – очікується «:»
87 «,» expected – очікується «,»
88 «(« expected – очікується «(»
89 «)» expected – очікується «)»
90 «=« expected – очікується «=»
91 «:=« expected – очікується «:=»
92 «[« or «(.» expected – очікується «[« або «(.»
93 «]» or «.)» expected – очікується «]» або «.)»
94 «.» expected – очікується «.»
95 «..» expected – очікується «..»
96 Too many variables – забагато змінних
97 Invalid FOR control variable – недопустима змінна циклу **FOR**
98 Integer variable expected – очікується змінна цілого типу
99 File types are not allowed here – тут не допускаються файлові типи
100 String length mismatch – невідповідність довжини рядка
101 Invalid ordering of fields – неправильний порядок полів
102 String constant expected – очікується константа рядкового типу
103 Integer or real variable expected – очікується змінна цілого або дійсного типу
104 Ordinal variable expected – очікується змінна зчисленого типу
105 INLINE error – помилка в операторі **INLINE**
106 Character expression expected – очікується вираз символьного типу

107 Too many relocation items – забагато переміщуваних елементів
108 Overflow in arithmetic operation – переповнення під час виконання арифметичної операції
109 No enclising For, While or Repeat statement – не включений в **For, While** або **Repeat**-конструкцію оператор
110 Cannot run a unit – модуль виконувати не можна
111 Compilation aborted – компіляція перервана
112 CASE constant out of range – константа оператора **CASE** виходить за допустимі межі
113 Error in statement – помилка в операторі
114 Cannot call an interrupt procedure – неможливо викликати напряму процедуру переривання
115 Must have an 8087 to compile this – для компіляції необхідна наявність сопроцесора 80x87
116 Must be in 8087 mode compile this – для компіляції необхідний режим використання 80x87
117 Targer address not found – адресу призначення не знайдено
118 Include files are not allowed here – в такій ситуації вкладення файлів неприпустиме
119 TP file format error – помилка формату файла .TP
120 NIL expected – очікується **NIL**
121 Invalid qualifier – неправильний кваліфікатор
122 Invalid variable reference – неправильне посилання на змінну
123 Too many symbols – забагато символів
124 Statement part too large – завеликий розділ операторів
125 Module has no debug information – в модулі немає інформації для налагодження
126 Files must be var parameters – файли повинні бути описані як VAR-параметри
127 Too many conditional symbols – забагато умовних символів
128 Misplaced condotional directive – пропущена умовна директива
129 ENDIF directive missing – пропущена директива **ENDIF**
130 Error in initial conditional defines – помилка в початкових умовних визначеннях
131 Header does not match previous definition – заголовок не відповідає попередньому визначенню
132 Critical disk error – критична помилка диска
133 Cannot evaluate this expression – неможливо обчислити даний вираз

134 Expression incorrectly terminated – некоректне завершення виразу
135 Invalid format specifier – неправильний специфікатор формату
136 Invalid inderect reference – неприпустиме непряме посилання
137 Structured variable are not allowed here – некоректне використання структурної змінної
138 Cannot evaluate without System unit – неможливо обчислити без модуля **System**
139 Cannot access this symbol – доступ до даного символу відсутній
140 Invalid floating-point operation – неприпустима операція з плаваючою комою
141 Cannot compile overlay to memory – не можна виконувати компіляцію оверлеїв в пам'ять
142 Procedual or function variable expected – очікується змінна-процедура або змінна-функція
143 Invalid procedure or function reference – неприпустиме посилання на процедуру або функцію
144 Cannot overlay this unit – цей модуль не може бути оверлейним
145 Too many nested scopes — забагато вкладених контекстів програми, де використано деяке ім'я
146 File access denied – доступ до файла заблокований **DOS**
147 Object type expected – очікується тип «об'єкт»
148 Local object types are not allowed – неприпустимі локальні описи типів об'єктів
149 VIRTUAL expected – необхідне слово **VIRTUAL**
150 Method identifier expected – очікується ідентифікатор методу
151 Virtual constructors are not allowed – віртуальні конструктори неприпустимі
152 Constructors identifier expected – очікується ідентифікатор конструктора
153 Descructor identifier expected – очікується ідентифікатор деструктора
154 Fail only allowed within constructors – виклик **Fail** припустимий тільки з конструктора

Для замовлення книг звертайтесь за адресою:

Шестопалов Євген Анатолійович, вул. Тургенєва, буд. 31, м. Шепетівка, Хмельницької обл., 30400

дом. тел. 8-03840-4-73-07, моб. тел. 8-066-283-66-18

E-mail: *aspekt@sh.km.ua*,

Ознайомитися з посібниками і зробити замовлення можна також з мого сайту *www.aspekt-edu.kiev.ua*

ДО КНИГ можна замовити ***ОДИН*** компакт-диск.

Караванова Т.П.

Інформатика

Основи алгоритмізації та програмування (процедурне програмування)

Навчальний посібник

Підписано до друку 20.09.07
Формат 60x84/16. Папір офсетний.
Гарнітура Times. Друк офсетний.
Ум. друк. аркуш 12.0
Зам.1725. Наклад 2500.

Видавець – Шестопалов Є.А.,
Свідоцтво про внесення до Державного реєстру
суб'єкта видавничої справи ДК № 2170 від 26.04.2005 р.
вул. Тургенєва, буд. 31, м. Шепетівка, Хмельницька обл., 30400,
Тел: (03840)-4-73-07, E-mail: *aspekt@sh.km.ua*

Шепетівська міжрайонна друкарня.
30400, м. Шепетівка, Старокостянтинівське шосе, 11
Свідоцтво ХЦ № 008 від 9.10.2000 р.
тел. (03840) 5-15-30