МІНІСТЕРСТВО ОХОРОНИ ЗДОРОВ'Я УКРАЇНИ
ЗАПОРІЗЬКИЙ ДЕРЖАВНИЙ МЕДИЧНИЙ УНІВЕРСИТЕТ
Кафедра медичної та фармацевтичної інформатики і новітніх технологій

Рижов О.А., Строітєлєва Н.І.

ФОРМАЛІЗАЦІЯ, АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ ФАРМАЦЕВТИЧНИХ ЗАДАЧ ЗАСОБАМИ TURBO-PASCAL

НАВЧАЛЬНИЙ ПОСІБНИК

для студентів фармацевтичного факультету
спеціальності 226 «Фармація, промислова фармація»

Запоріжжя
2019

УДК 004.4:004.94: 615](075.8)
Р 93

Затверджено на засіданні Центральної методичної Ради ЗДМУ та рекомендовано для використання в освітньомку процесі (протокол № 5 від «23» травня 2019 р.)

Розробники:

О.А Рижов – д-р фарм.н., проф., завідувач кафедри медичної та фармацевтичної інформатики і НТ ЗДМУ
Н.І. Строітєлєва– к.ф.-м.н., доцент кафедри медичної та фармацевтичної інформатики і НТ ЗДМУ

Рецензенти:

І.А. Арутюнян - зав.кафедри промислового та цивільного будівництва інженерного інституту ЗНУ, д.т.н., проф.
Г.П.Коломоєць - директор будівництва інженерного інституту ЗНУ, к.ф.-м.н.,
доцент

Р93

Рижов О.А.

Формалізація, алгоритмізація та програмування фармацевтичних задач засобами Turbo-Pascal : навчальний посібник для студентів фармацевтичного факультету спеціальності 226 «Фармація, промислова фармація» / Рижов О.А., Строітєлєва Н.І. – Запоріжжя: ЗДМУ, 2019. – 141 с.

Навчальний посібник «Формалізація, алгоритмізація та програмування фармацевтичних задач засобами Turbo-Pascal» призначений для отримання практичних навичок зі складання алгоритмів фармацевтичних задач та їх програмування засобами мови Pascal. Посібник розрахований на студентів фармацевтичного факультету очної та заочної форм навчання для проведення практичних занять з навчальної дисципліни «Інформаційні технології у фармації»

ЗМІСТ

ВСТУП

Міністерство охорони здоров'я України проводить роботу зі створення повної бази медико-технологічних документів зі стандартизації медичної допомоги. Нова база повина буди представлена трьома типами документів: уніфіковані клінічні протоколи медичної допомоги, стандарти медичної допомоги і адаптовані клінічні керівництва. Невід'ємною частиною цих документів є алгоритми, зокрема алгоритми дій лікаря при виконанні діагностики і лікуванні, а також алгоритми проведення медичних маніпуляцій. Для ефективного використання цих документів в реальних умовах, недостатньо уявити опис алгоритмів в текстовому вигляді. Необхідно візуалізувати алгоритми, дати їх графічне уявлення за допомогою схем та програм, які зчитує комп’ютер. Це полегшить розуміння алгоритмів і дозволить лікарю швидко визначати, які дії слід виконувати на кожному ця-пе роботи з пацієнтом. Точна і зрозуміла схема алгоритму повинна сприяти зменшенню частоти лікарських помилок.

Створити такі програми та алгоритми дозволяє система програмування Turbo Pascal. Вона розроблена американською фірмою Borland і досі залишається однією з найпоширеніших систем. Цьому сприяє простота освоєння мови, можливість створення структурованих програм для вирішення як обчислювальних завдань, так і завдань, пов'язаних з обробкою складних структур даних. Мова програмування високого рівня Pascal використовується при розробці операційних систем і систем управління базами даних. Інструментальні засоби, що з'явилися в даний час для розробки програм, такі як Borland Pascal, Delphi, що працюють в Windows, грунтуються на Турбо Паскалі і розвивають його ідеї.

Даний навчальний посібник написаний в рамках вивчення курсу «Інформаційні технорлогії у фармації» студентами спеціальності 226 «Фармація».

1. ОСНОВИ АЛГОРИТМІЗАЦІЇ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

1.1. Етапи розробки програм на персональному комп'ютері

Формалізація - процес подання інформації про об'єкт у вигляді алгоритму. В результаті аналізу завдання визначається специфіка даних, вводиться система умовних позначень, встановлюється приналежність її до одного з класів задач (наприклад, математичні, фізичні, медичні тощо). Якщо певні аспекти розв'язуваної задачі можна виразити в термінах якої-небудь формальної моделі (певної структури, яка використовується для представлення даних), то це, безумовно, необхідно зробити, оскільки в цьому випадку в рамках формальної моделі можна дізнатися, чи існують методи і алгоритми вирішення поставленого завдання. Навіть якщо вони не існують, то використання коштів і властивостей формальної моделі допоможе в побудові рішення задачі. Формалізована медико-біологічна задача повинна бути алгоритмізованою. Під алгоритмізацією розуміють метод опису систем або процесів шляхом створення алгоритмів їх функціонування.

З розвитком обчислювальної техніки і теорії програмування зростає необхідність побудови нових алгоритмів, змінюються способи їх побудови, способи запису алгоритмів мовою, зрозумілою виконавцю. Особливий тип виконавця алгоритмів - комп'ютер, тому необхідно створювати спеціальні засоби, що дозволяють, з одного боку, розробнику в зручному вигляді записувати алгоритми, а з іншого - дають комп'ютеру можливість розуміти написане. Такими засобами є мови програмування або алгоритмічні мови.

Програма **–** це логічно впорядкована послідовність команд, необхідних для управління комп'ютером, тому програмування зводиться до створення послідовності команд, необхідного для вирішення певного завдання.

Процес розробки нових програм для ЕОМ включає:

1. постановку завдання;
2. створення алгоритму її рішення;
3. реалізацію алгоритму на ЕОМ у вигляді програми ;

4. відладку програми.

Розглянемо по черзі всі ці етапи. Постановка завдання полягає в чіткому формулюванні цілей роботи. Необхідно чітко визначити, що є початковими даними, що потрібно одержати як результат, яким повинен бути інтерфейс програми (тобто яким чином здійснюватиметься діалог з користувачем) і т.д.

Алгоритм – докладний чіткий опис послідовності операцій, які потрібно виконати для вирішення завдання. Розробку алгоритму можна порівняти з прокладенням трамвайних колій, при якому потрібно передбачити систему стрілок, розворотів так, щоб за будь-яких умов трамваї могли по прокладених шляхах дійти від початкового пункту маршруту до кінцевого. Про програму, що виконує дії, наказані алгоритмом, говорять, що вона реалізує даний алгоритм на ЕОМ.

Графічна інтерпретація алгоритму називається ***блок-схемою***. Як приклад розглянемо блок-схему простого і добре всім відомого алгоритму переходу вулиці через перехрестя, обладнане світлофором (рис.1.1).

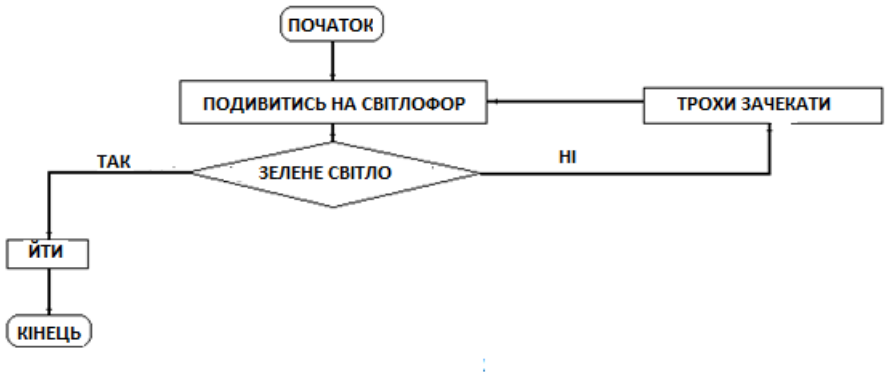


Рисунок 1.1 – Приклад графічної інтерпретації алгоритму

Сучасний стан програмування ставить такі вимоги до алгоритмів:

1. Відсутність помилок;
2. Однозначність, тобто чітке розпорядження, що і як робити в кожній конкретній ситуації. Ніяких неоднозначностей ("можна зробити так, а можна і так...") бути не повинне. Один з пунктів розглянутого вище алгоритму переходу вулиці звучить неоднозначно – "небагато почекати". Зрозуміло, що даний

алгоритм орієнтований на людину, а людина зрозуміє, що означає слово "небагато", правда кожен по-своєму. Для комп'ютера поняття "трохи" не існує, тому при створенні машинно-орієнтованих алгоритмів потрібно указувати конкретні величини, наприклад "почекати 3 секунди".

3. Універсальність, тобто застосовність даного алгоритму до рішення будь-якої задачі даного типа. Дана вимога економічна. Розробка серйозної програми це дуже складний, тривалий і трудомісткий процес, і окупиться він тільки тоді, коли створена в результаті програма використовуватиметься багато разів. Писати програми, які будуть використані тільки один раз сенсу немає. Виключенням можуть бути тільки якісь особливі випадки і навчання програмуванню.

4. Результативність, тобто відсутність зациклень. Будь-яка програма повинна завжди приводити до результату, навіть якщо цим результатом буде аварійне повідомлення. Світлофор не працює, отже створюється аварійна ситуація, відбудеться зациклення або матимуть місце які-небудь інші непередбачувані результати. Тоді, у ряді випадків програми "зависають", або зациклюються, як в даній ситуації. Вийти із замкнутого круга, що утворився, можна тільки примусовим перериванням роботи програми, наприклад, шляхом перезавантаження комп'ютера.

Наступним кроком після створення алгоритму є написання реалізуючої його програми. Основна складність тут полягає в тому, що програма є набором двійкових кодів-нулів і одиниць. Алгоритм же формулюється на природному людській мові – російській, англійській, німецькій, арабській і т.д. У зв'язку з цим в даний процес вводиться проміжний етап – розробка тексту програми (рис.1.2).

Мова програмування – штучна мова, що є проміжною при переході від природної людської мови до машинних двійкових кодів. Річ у тому, що в комп'ютерній технології використовується т.з. двійкове представлення інформації у вигляді 1 і 0. Комп'ютером можна управляти, знаючи код, що складається з 1 і 0,

який визначає виконання певної команди. Для нормального спілкування людини з ЕОМ необхідне існування посередника, яким і є мова програмування.

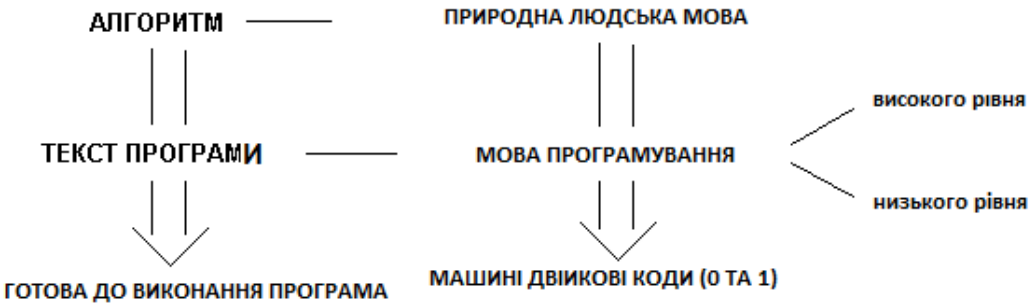


Рисунок 1.2 – Місце мови програмування в процесі створення програми

Така мова використовує слова, подібні тим, які використовує людина, але вони мають строго певний синтаксис (порядок слів, граматика, орфографія), що не залишає неоднозначності, поширеної в звичайній мові. Комп'ютер може переводити слова мови програмування в мову 1и 0, а оскільки така мова подібна розмовній мові, людина може вивчити його і використовувати ефективніше, ніж програмувати за допомогою 1 і 0.

Останній етап – відладка програми – це виправлення в ній помилок і ретельне її тестування. Існують 2 типу помилок:

- синтаксичні – їх легше виявити, вони пов'язані з командами і пунктуацією.
- логічні – виникають, коли з синтаксично коректною програмою намагаються виконати нездійсненні дії.

При тестуванні програми важливо перевірити її працездатність як можна в більшому числі ситуацій, наприклад, при різних варіантах початкових даних. Буває, що в 1000 випадках програма спрацює нормально, а на 1001-й раз виявиться помилка. При написанні серйозних програмних продуктів для повнішого їх тестування фірми-розробники часто поширюють їх пробні версії (версії бети) серед

як можна більшого числа користувачів, які повідомляють у фірму про виявлені помилки, що дозволяє виправити їх в остаточних версіях програмних продуктів.

1.2.Форми запису алгоритмів

Існують безліч різних форм запису алгоритмів. Це пов'язано з тим, що кожен виконавець алгоритмів "розуміє" лише такий алгоритм, який записаний на його "мові" і за його правилами. Умовно виділяють наступні форми запису алгоритмів:

1. Словесно-покрокова (текстова).
2. Таблична.
3. Графічна форма запису (блок-схема).
4. Запис на алгоритмічній мові.

Запис всякого алгоритму починається із заголовка.

У разі ***словесно-покрокової форми*** алгоритм записується у вигляді пронумерованих етапів його виконання. Наприклад розглянемо алгоритм складання двох чисел (a і b). Він має вигляд:

1. Запитати, чому рівне число a.
2. Запитати, чому рівне число b.
3. Скласти a і b, результат привласнити c.
4. Повідомити результат c.

Таблична форма запису – це запис алгоритму у вигляді таблиці. Використовувані таблиці можуть бути різними. Для прикладу використовуватимемо спрощену форму. Порядок складання табличних алгоритмів наступний :

1. Переписати вираз так, як допустимо в інформатиці.
2. Визначити порядок дій.
3. Ввести позначення проміжних результатів.
4. Занести одержані дії в таблицю.

Приклад:

№ дії	Дія	*Алгоритм обчислення R=2a +3b.*		результат
		1	2	
1	*	2	a	до
2	*	3	b	u
3	+	до	u	R

Запис на алгоритмічній мові – це запис алгоритму на спеціальній мові (на мові програмування). Він здійснюється, строго слідуючи правилам тієї або іншої алгоритмічної мови. Заголовок включає назву алгоритму, імена початкових даних (це величини, без яких виконати алгоритм неможливо) і імена результатів (це величини, значення яких обчислюються в алгоритмі). Для вказівки почала і кінця алгоритму використовуються службові слова ***початок*** і ***кінець***. Між ними записують одну або декілька команд алгоритму, їх називають тіло алгоритму.

1.3.Графічна форма запису алгоритму

Алгоритм записується у вигляді схеми, що складається з блоків (геометричних фігур) з розміщеними в них діями (табл. 1.1). Блоки з'єднуються стрілками і показують структуру всього алгоритму. Алгоритм у вигляді блок-схеми починається блоком «почало» і закінчується блоком «кінець».

При складанні блок-схеми алгоритму спочатку виділяють початкові дані (всі змінні величини після знаку рівності і в умові) і результат (величини які необхідно знайти). Якщо в завданні маються на увазі, але не указуються імена величин, то вони позначаються самостійно. За відсутності початкових даних блок введення не пишеться. У один блок можна помістити одну дію.

На рис. 1.3 наведений приклад скаладння лінійного алгоритму проведення медичних маніпуляцій при дослідженні звуків дихання, що проводяться за допомогою комплекса «КоРа – 03 М1». Наведений алгоритм демонструє, що трансформація звукових образів у візуальні дозволяє ефективно та достовірно

об’єктивізувати аускультаційні показники, що характеризують певний вид бронхолегеневого захворювання.

Рисунок 1.3 – Приклад фрагменту лінійного алгоритму

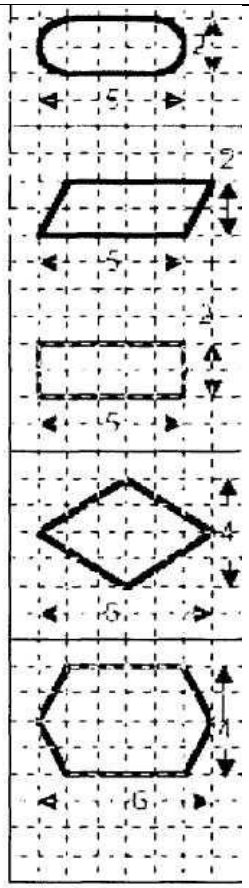
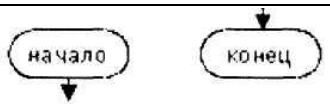
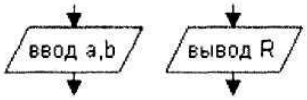
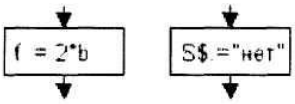
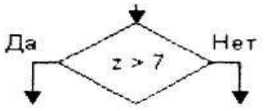
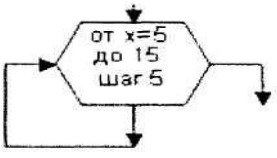
Існують наступні види алгоритмів :

- лінійний
- що розгалужується
- циклічний
- комбінований.

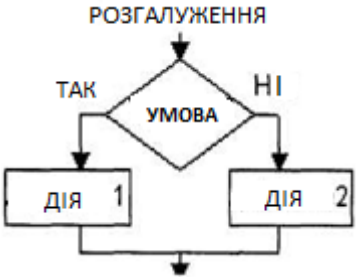
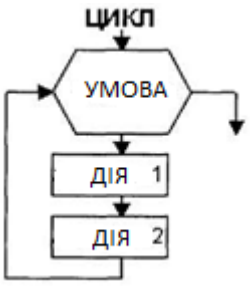
При визначенні виду алгоритму користуються ключовими словами.

Алгоритм, який містить декілька структур одночасно, називається ***комбінованим.*** Розглянемо приклад комбінованого алгоритму, приведений на рис. 1.4.

Таблиця 1.1 – Основні блоки, що беруть участь в створенні блок-схем

Вид блоку	Назва / призначення	Приклади запису
	**блок почала / кінця алгоритму** позначає початок або кінець алгоритму	
	блок введення / виведення служить для введення початкових даних і виведення результатів	
	блок дії служить для запису команди привласнення	
	блок логічної умови служить для організації галуження в алгоритмі	
	блок циклу служить для організації **циклів** в алгоритмі	

Таблиця 1.2 – Види алгоритмів

Вид алгоритму	Ключові слова	Структура
Алгоритм, в якому є структура «Проходження» називається **ЛІНІЙНИМ.** ***Проходження*** – це розташування дій один за одним.	*Ключових слів немає.*	
Алгоритм, у якому є структура «Галуження» називається **СТРУКТУРА ЩО РОЗГАЛУЖУЄТЬСЯ.** ***Галуження*** **–** це вибір дії залежно від виконання будь-якої умови.	*якщо...то...інакше.. ;при ...(у значенні якщо...).*	
Алгоритм, в якому є структура «Цикл» називається **ЦИКЛІЧНИМ.** ***Цикл*** – це неодноразове повторення яких-небудь дій	*Від...до...; ... раз; поки...; якщо... (у значенні поки...).*	

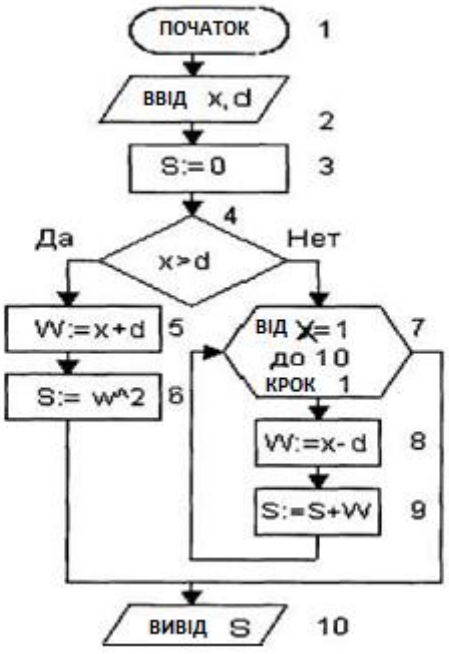


Рисунок 1.4 – Приклад комбінованого алгоритму

Визначимо, якими блоками реалізована його структура. Структура «Проходження» реалізована в блоках 1-3, 5, 6, 8 - 11, структура «Галуження» – в блоках 4 -9, структура «Цикл» – в блоках 7-9.

1.4. Складання алгоритмів, що розгалужуються

Галуження в алгоритмах дозволяє виконати дію (або серію дій) залежно від виконання або невиконання якої-небудь умови. Умова є рядком, що містить операцію порівняння з використанням знаків **< > = .**

Наприклад:

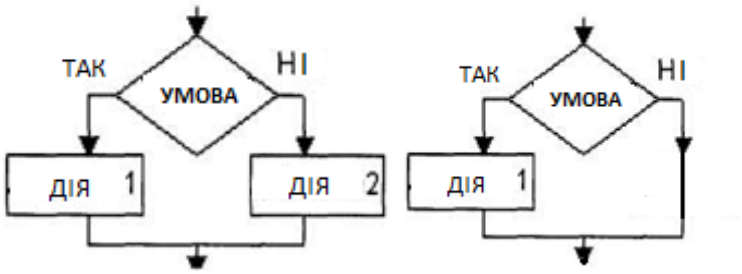
x>5; sl<-15.5; d$="так" ; j<>4 (не рівно);

Z3 > = 3 (більше або рівно);

t < = 0 (менше або рівно).

Якщо умова дотримується**,** то виконуються дії, розташовані в гілці під назвою "Так". У разі недотримання умови будуть виконані дії, розташовані в гілці "Ні".

Структура «Галуження» в алгоритмах, що розгалужуються, може бути представлене в двох формах: повної або неповної (рис. 1.5). Неповна форма галуження відрізняється від повної тим, що в одній з гілок неповної форми дії відсутні. У такому алгоритмі відповідно до умови або будуть виконані дії, наявні в гілці, або почнуть відразу виконуватися дії, розташовані після галуження.



а) б)

Рисунок 1.5 – Блок-схема повного (а) і неповного (б) галуження в алгоритмі

На алгоритмічній мові структура «Галуження» записується так, як показано в табл.1.3. Службове слово ***якщо*** позначає початок галуження, а ***все*** - кінець галуження.

Таблиця 1.3 –Запис алгоритму повного і неповного галуження на алгоритмічній мові

У повній формі	**У неповній (скороченої) формі**
якщо умова ***то*** дія 1 ***інакше*** дія 2 ***все***	***якщо*** умова ***то*** дія ***все***
залежно від умови в рядку якщо виконується тільки одна з дій (або група дій) розташованих або в рядку ***то*** (умову дотримано), або в рядку ***інакше*** (умову не дотримано).	в цьому випадку виконається дія (або група дій) розташована в рядку ***то*** тільки при дотриманні умови Якщо ж умова не дотримується, то виконавець перейде до виконання дій, наступних за службовим словом ***все***.

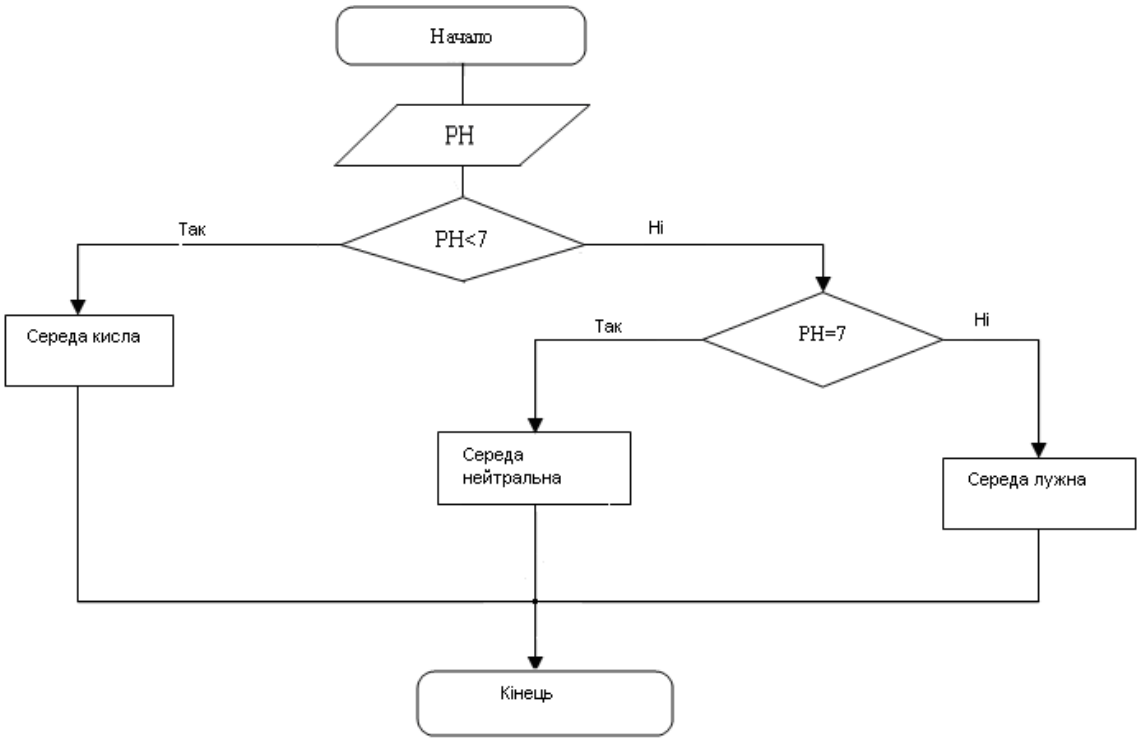


Рисунок А.А – Приклад складання алгоритму, що розгалужується

1.5.Складання циклічних алгоритмів

Структура «Цикл» використовується при складанні алгоритмів, в яких необхідно багато разів повторювати які-небудь дії.

1.5.1.Цикл з параметром

Для організації циклу з параметром вводиться величина **(*лічильник*),** яка міняє своє значення від ***початкового*** до ***кінцевого*** з певним ***кроком*.** Крок рівний різниці між наступним і попереднім значенням величини. Якщо при виконанні алгоритму повинен вийти ряд відповідей, то блок висновку поміщається усередині циклу.

Всі дії, розміщені усередині циклу, називаються тілом циклу. Тіло циклу виконується стільки разів, скільки різних значень прийме параметр в заданих межах.

Наприклад :

від X=10 до 13 крок 1 тобто X прийматиме значення 10,11,12,13 (рис.1.6);

від R=20 до 14 крок -2 тобто R прийматиме значення рівні 20,18,16,14.

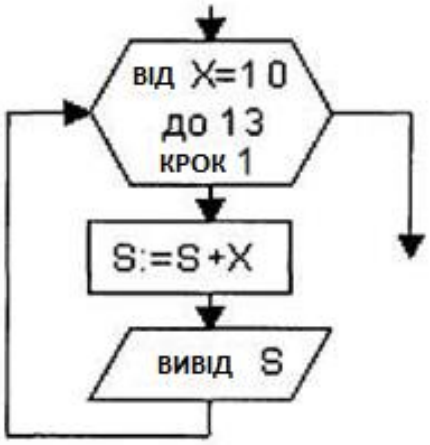


Рисунок 1.6 – Блок – схема циклу з параметром

Всі дії, розміщені усередині циклу, називаються ***тілом*** циклу. Тіло циклу виконується стільки разів, скільки різних значень прийме параметр в заданих межах.

Для організації циклу так само можна використовувати блок логічної умови.

1.5.2. Цикл з передумовою

Тіло циклу з передумовою розміщується після перевірки умови його закінчення. Цикл з передумовою виконується до тих пір, поки умова істинно (True), інакше цикл завершується. Цикл може не виконатися жодного разу, якщо при першій же перевірці умова виявиться помилковою (False).

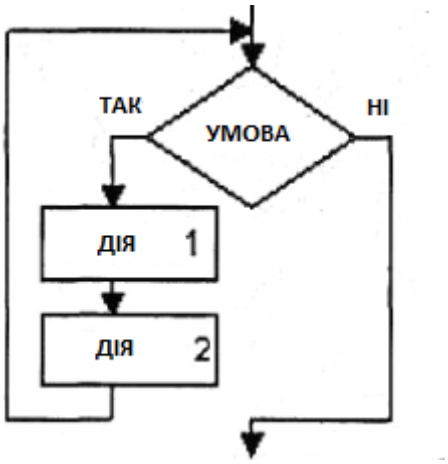


Рисунок 1.7 – Блок – схема циклу з передумовою

1.5.3.Цикл з післяумовою

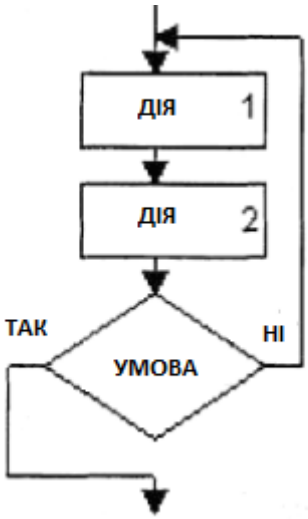


Рисунок 1.8 – Блок – схема циклу з післяумовою

Тіло циклу з післяумовою розміщується до перевірки умови його закінчення, тому такий цикл обов'язково виконається хоч би один раз. Цикл з післяумовою виконується до тих пір, поки умова буде помилковою (False). Якщо умова виявиться істинною (True), цикл більше не виконуватиметься (рис.1.8).

1.6 Практичні роботи до розділу

Практична робота №1

Алгоритмізація обчислювальних структур, що розгалужуються

Завдання 1: скласти блок-схему алгоритму медичного процесу діагностики туберкульозу органів дихання у дітей згідно його текстової інструкції:

1. Провести діагностику пацієнта.
2. Якщо діагностика підтвердила діагноз «туберкульоз», пацієнтові призначити відповідне лікування, потім провести період реабілітації та повторну діагностику. Потім перевести пацієнта на діспансерне спостереження.
3. Якщо в результаті першої діагностики діагноз «Туберкульоз» непідтверджений, пацієнт вважається здоровим і переводиться на диспансерне спостереження.

Приклад виконання завдання 1

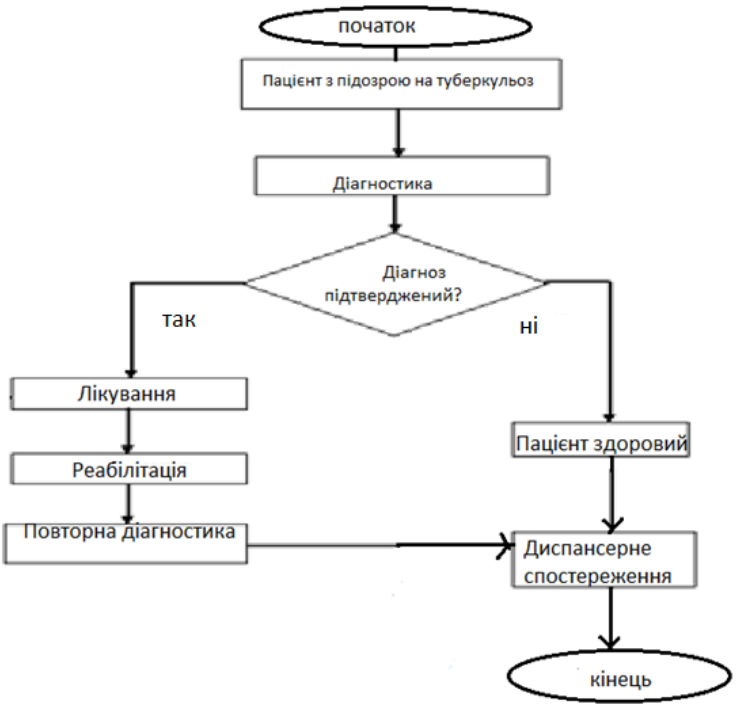


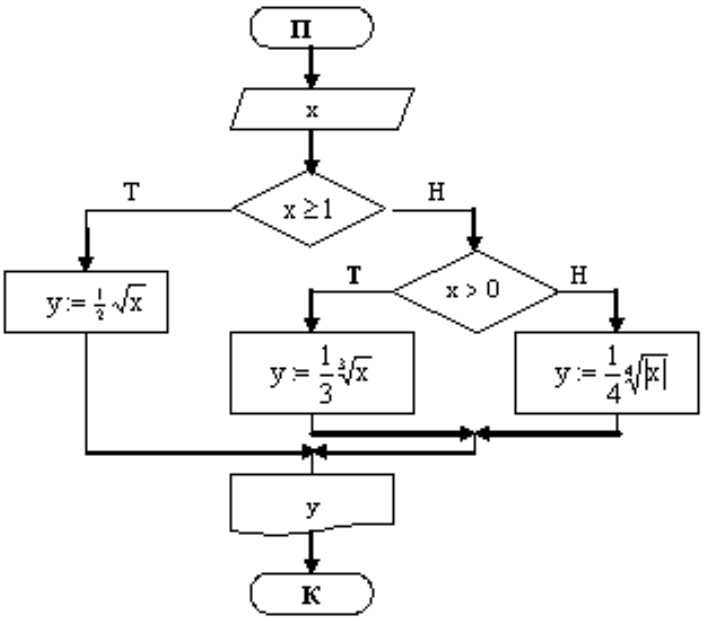
Рисунок 1.9 - Діагностика туберкульозу у дітей

Завдання 2: скласти блок-схему алгоритму обчислення значення функції *y = f(x):*

$$y=\begin{cases}\frac{1}{2}\sqrt{x}, & \textit{якщо } x\geq 1;\\ \frac{1}{3}\sqrt[3]{x}, & \textit{якщо } 0\prec x\prec 1;\\ \frac{1}{4}\sqrt[4]{|x|}, & \textit{якщо } x\leq 0;\end{cases}$$

При рішенні даної задачі можливі варіанти програмування з вкладеною структурою.

Приклад виконання завдання 2



Завдання для самостійної роботи

1. Скласти блок-схему алгоритму медичного процесу діагностики кліщового бореліозу згідно його текстової інструкції:

1) якщо дігностика виявила мігруючу ерітему, то встановити діагноз – еритемне захворювання Лайма.

2) якщо дігностика не виявила мігруючу ерітему, провести обстеження на загальноінфекційний синдром та регіональний лімфаденіт;

3) якщо діагностика позитивна, провести епіданамнез;

4) якщо епіданамнез позитивний, перевірити значення РНІФ, якщо він перевищує 1,64, то встановити діагноз – безеритемне захворювання Лайма.

2. Скласти блок-схему алгоритму медичного процесу обстеження хворого з гемодіалізом, якщо час кровотечі не первищує 3 хвилини, згідно його текстової інструкції:

1) провести дослідження плазмового компонента гемостаза та визначення АЧТВ, ПВ, ТВ. Якщо будь-який з цих параметрів не співпадає з нормою, діагнозувати порушення плазмового компонета;

2) в іншому випадку визначити розчинність фібринового згустку в сечовині; якщо вона в нормі, діагнозувати дефект судинного компонета гемостазу;

3) якщо розчинність прискорена, діагностувати -дефект XIII фактора.

3. Скласти блок-схему алгоритму медичного процесу обстеження хворого з гемодіалізом, якщо час кровотечі первищує 3 хвилини, згідно його текстової інструкції:

1) визначити кількість тромбоцитів: якщо вона знижена, діагностувати захворювання, що викликають тромбоцитопенію;

2) якщо кількість тромбоцитів підвищена, діагностувати захворювання, що викликають тромбоцитоз;

3) якщо кількість тромбоцитів у нормі, дослідити функціональні властиворсті тромбоцитів. Якщо ці показники змінені, діагностувати захворювання, що пов’язані тромбоцитопатією.

4) В протилежному випадку діагностувати дефект судинного компоненту гемостаза.

4. Скласти блок-схему алгоритму медичного процесу обстеження хворого з клініко-лабораторними ознаками БЦЖиту, згідно його текстової інструкції:

1) провести бактеріологічне та морфологічне дослідження оперійного матеріалу;

2) якщо туберкульозне запалення доказане та культура отримана, провести молекулярно- генетичну ідентифікацію мікробактерій, а потім ідентифікацію M.bovis DCG і встановити діагноз – БЦЖит;

3) якщо туберкульозне запалення доказане, але культура не отримана, провести діаскін- тест;

4) якщо результати тесту позитивні, діагностувати туберкульоз, в іншому випадку встановити діагноз – БЦЖит.

5. Скласти блок-схему алгоритму медичного процесу обстеження хворого з підозрою на мастоідит, згідно його текстової інструкції:

1) якщо після діагностики діагноз «мастоідит» не підтверджений, продовжити діагностичний пошук;

2) у протилежному випадку перевірити наявність показань до хірургічного втручання. Якщо є такі підстави, виконати хірургічне лікування та провети потім консервативну терапію;

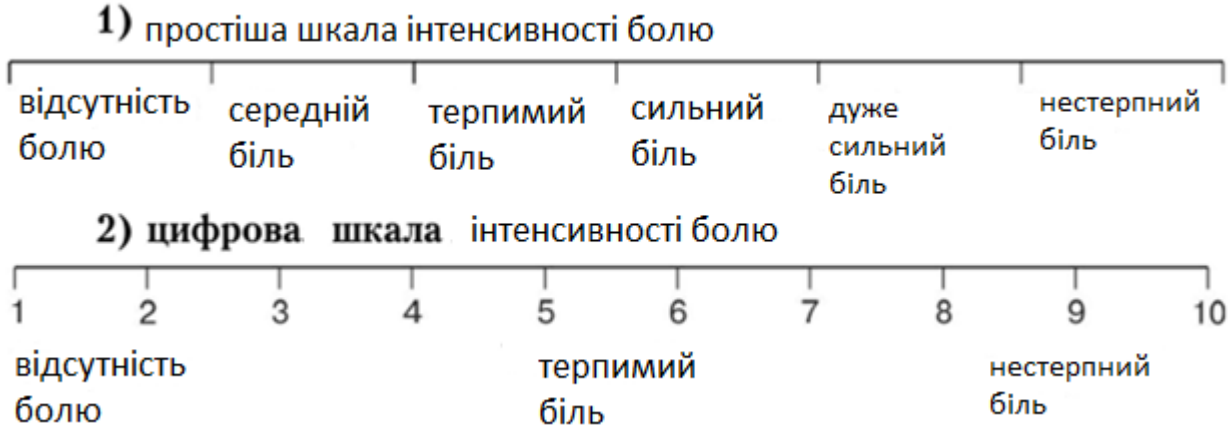
3) якщо підстав до хірургічного втручання немає, провести консервативну терапію та згодом перевірити її ефективність. Якщо ефективність низька, повернутися до хірургічного лікування пацієнта.

6. Скласти блок-схему алгоритму медичного процесу обстеження хворого з генералізованою лімфаденопатією, згідно його текстової інструкції:

1) провести аналіз змін у крові пацієнта, якщо вони присутні, провести діагностичний пошук згідно з виявленими змінами. Якщо після цього діагнозу все ще немає, зробити біопсію лімфатичного вузла;

2) якщо зміни крові намає, але існують доадткові ознаки, провести діагностичний пошук згідно з виявленими змінами. Якщо після цього діагнозу все ще немає, зробити біопсію лімфатичного вузла.

7. Скласти блок-схему алгоритму медичного процесу діагностики інтенсивності болю хворого згідно графічної інструкції:



8. Скласти блок-схему алгоритму медичного процесу призначення серцевого препарату «Корглікона»в залежності від віку пацієнта: 1) до 2 років не призначається; 2) 2-6 років по 0,1-0,5 мг; 3) 6-12 років по 0,5-0,75 мг; 4) после 12 років – по 0.75-1 мг.

9. Скласти блок-схему алгоритму медичного процесу визначення жінок з факторами ризику остеопорозу згідно його текстової інструкції:

1) якщо у пацієнтки вже були переломи при мінімальній травмі, треба призначити лікування остеопорозу;
2) у протилежному випадку треба оцінити вік пацієнтки: якщо її вік перевищує 65 років, їй також треба призначити лікування остеопорозу;
3) якщо вік пацєнтки перебільшує 65 років, їй призначають вимірювання МПК та прийом глюкокортикоїдів.

10. Скласти структурну схему алгоритму для обчислення об’єму води (ОВ), що містить організм, для дорослого пацієнта в залежності від ваги, полу, на підставі наступних умов: 1) для чоловіків ОВ = вага * 0,8, 2) для жінок ОВ = вага *0,75.

Практична робота №2

Алгоритмізація циклічних обчислювальних структур

Завдання 1: побудувати структурну схему алгоритму для перекладу температури з градусів за шкалою Цельсія (C) в градуси за шкалою Фаренгейта (F) для значень від 20° C до 32° C з кроком 0,5° C за формулою F: = 1,8 * C + 32.

Приклад виконання завдання

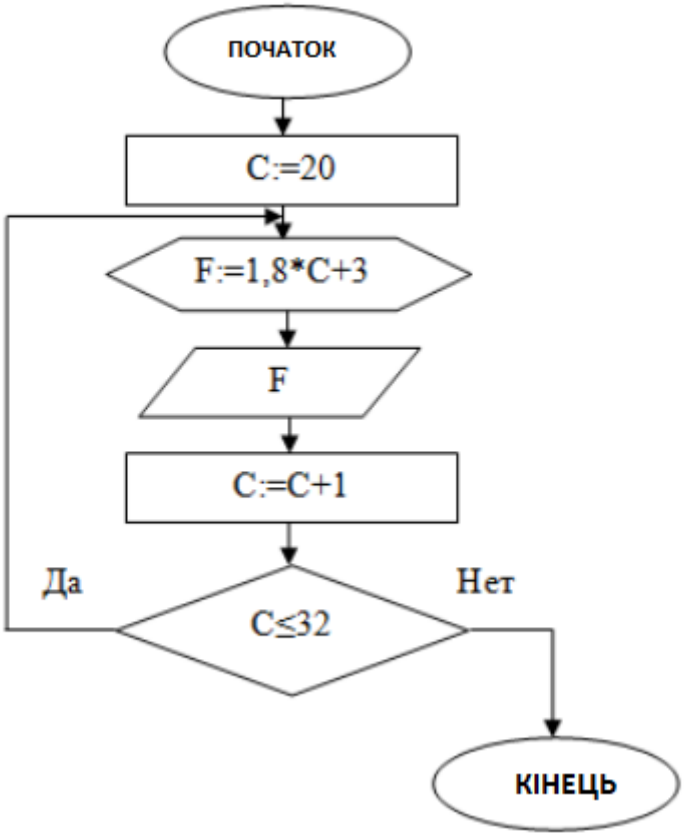


Рисунок 1.10 - Приклад алгоритму циклу з парметром

Завдання 2: скласти алгоритм табулювання функції з однією змінною, для цього обчислити таблицю значень функції:

$$y(x) = \frac{\ln|x|}{a^2 + b^2}$$

для x, що змінюється в інтервалі [-0.5, 2.5] з кроком 0,1.

У даному завданні змінна є змінною циклу, що управляє, a та b – задані дійсні числа.

Приклад виконання завдання 2

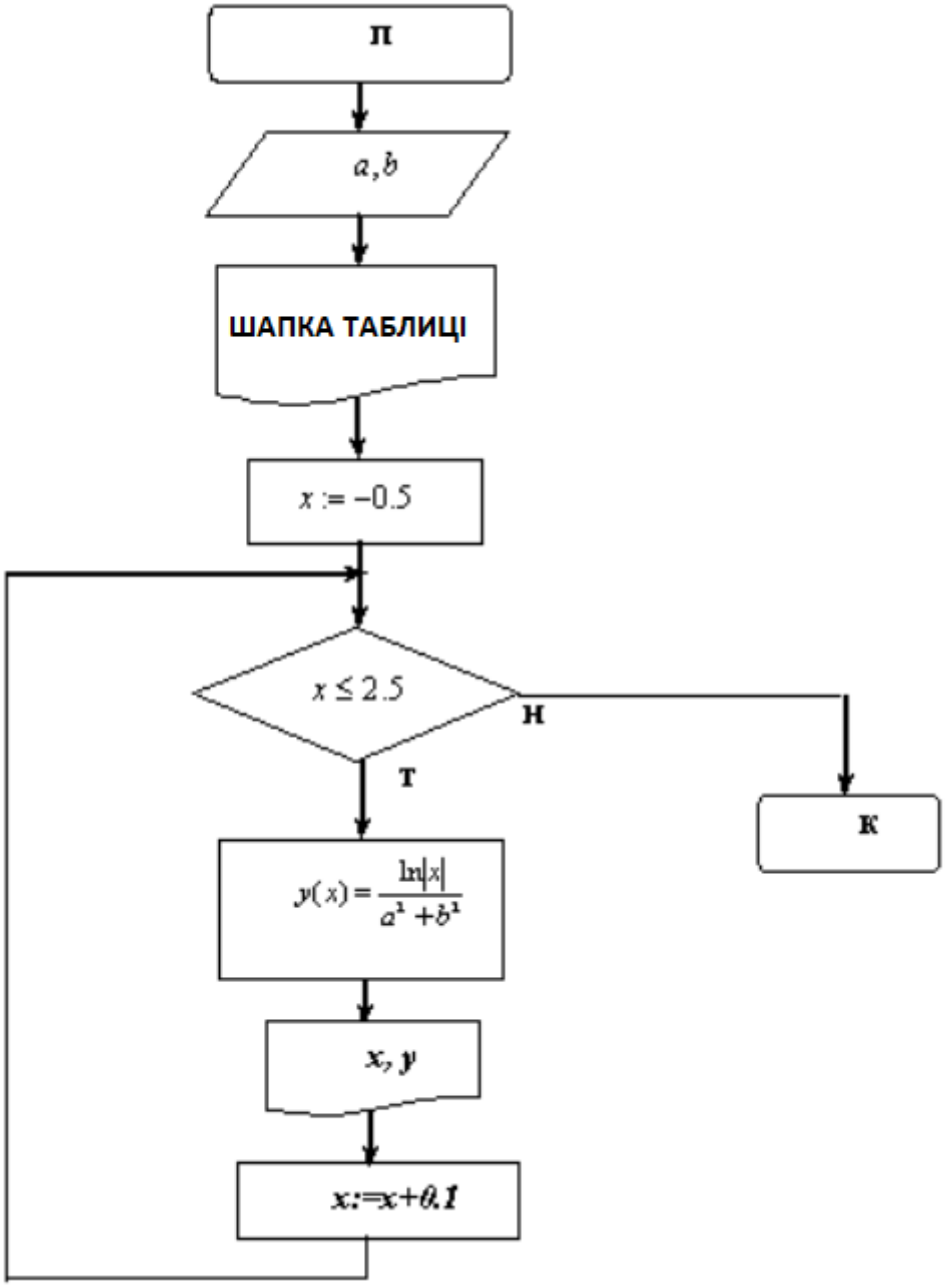


Рисунок 1.11 - Приклад алгоритму циклу з передумовою

Завдання для самостійної роботи

1. Скласти блок-схему циклічного алгоритму надання невідкладної медичної допомоги дитині з анафілактичним шоком згідно його текстової інструкції:

1) негайно припинити подальше надходження алергену в організм

2) дитину укласти на бік, щоб уникнути асфіксії в результаті аспірацій блювотних мас, западання язика;
3) при відсутності блювоти пацієнта укласти на спину з піднятими нижніми кінцівками;
4) забезпечити доступ свіжого повітря, прохідність дихальних путів. Зігріти пацієнта, обкласти грілками;
5) підшкірно ввести 0,1% розчин адреналіну 0,05-0,1 мл / рік життя, але не більше 1 мл. Введення препарату повторити через 15-20 хв.
6) швидка госпіталізація дитини.

2. Побудувати структурну схему циклічного алгоритму для визначення тиску крові в аорті за формулою: P = Po * e -t / (x k) в інтервалі 0 t 1 (c) з кроком Δt = 0,1 (c), Po - початкове значення тиску крові, x - гідравлічний опір аорти, k - коефіцієнт еластичності аорти.
3. Побудувати структурну схему циклічного алгоритму для визначення скорочення м'язи, відповідно до рівняння Релея: Y = b * t * e - (k * t * t) / 2, де t – інтервал часу від 0 до 15 хвилин з кроком 1 хвилина, b - постійна, k – постійна.

4. Скласти циклічний алгоритм медичних дій по введенню преднізолону (0,1-0,2 мл / кг) або гідрокортизону (4-8 мг / кг) кожні 4-6 годин.

5. Скласти циклічний алгоритм медичних дій по визначенню показника ШОЕ у пацієнтів різної статі з метою встановлення відхилень показника від норми для подальшого медичного обстеження.

6. Скласти циклічний алгоритм дій визначенню віку пацієнтів (число) під час заповнення медичної карти. Дати для цього числа найменування «рік», «року» або «років»: наприклад, 21 рік, 44 роки, 65 років і т.д.

7. Динаміка концентрації амфітаміну у плазмі крові за певного рівня кислотності PH описується рівнянням

$$C_{амфетамина}(t) = e^{\frac{-0,693 \cdot t}{7 \cdot ph - 37,5}},$$

де C – концентрація препарату; e- час після введення препарату; PH = 6.5 - кислотність. За умови, що пацієнтові було введено амфетамін у час t = 0, визначте, як змінювалась концентрація в плазмі крові амфетаміну кожні 10 хвилин протягом 2 годин.

8. Динаміка концентрації метілфенідату гідрохлориду у плазмі крові за певного рівня кислотності PH описується рівнянням

$$C_{метилфенидата}(t) = e^{-0{,}365 \cdot t}$$

де C – концентрація препарату; e- час після введення препарату; PH = 6.5 - кислотність. За умови, що пацієнтові було введено препарат у час t = 0, визначте, як змінювалась концентрація в плазмі крові метілфенідату кожні 15 хвилин протягом 3 годин.

9. Даний каскад з N послідовних нейронів, що передають сінаптичний сигнал. Сигнал передається від одного нейрона іншому з однаковою затримкою у часі $\tau = 0{,}5$с. В цьому випадку швидкість розповсюдження сигналу R як функція від часу t (у секундах) задається формулою:

$$R(t) = \frac{1}{\tau \cdot 10!} \cdot \left(\frac{t}{\tau}\right)^N \cdot e^{\frac{-t}{\tau}}$$

Визначте , як змінюється швидкість розровсюдження сигналу кожну 1 с.

10. Зростання кількості бактерій у популяції описується рівнянням:

$$f(t) = \frac{a}{1 + b \cdot e^{-a \cdot t}},$$

де f – кількість бактерій у популяції (в тисячах), t – час розвитку популяції (дні), a – гранична кількість бактерій у популяції (в тисячах), b – параметр, пов’язаний зі швидкістю розмноження бактерій. Визначте, як змінюється популяція кожен день на протязі 10 днів, якщо a=1,2, b=15,33.

1.6 Контрольні питання до розділу 1

1. Що таке мова програмування ?
2. Що таке алгоритм?
3. Що виконує програма – транслятор ?
4. Що таке відладка програми ?
5. Що таке блок – схема ?
6. Назвіть основні етапи створення програми на ЕОМ.
7. Назвіть та опишіть різні види алгоритмів.
8. З якого блоку починається будь-яка блок-схема алгоритму ?
9. Яким блоком закінчується алгоритм у вигляді блок-схеми ?
10. Для чого в блок-схемі служить блок дії ?
11. Який блок служить для організації галуження в блок-схемі?
12. Який блок у блок-схемі служить для введення початкових даних і виведення результатів?
13. Як називається розташування дій в блок-схемі один за одним?
14. Як називається алгоритм, який містить декілька структур одночасно?
15. Що таке структура «Цикл»?
16. Що таке структура «Галуження»?
17. Що є умова в блоці логічної умови?
18. Чим відрізняється неповна форма галуження в блок-схемі від повної?
19. Що таке тіло циклу?
20. Для організації чого вводиться величина – ЛІЧИЛЬНИК?
21. Чим характеризується цикл з післяумовою?
22. Чим характеризується цикл з передумовою?
23. Як виконується тіло циклу з параметром?

2. ОСНОВНІ ПОНЯТТЯ І КОНСТРУКЦІЇ АЛГОРИТМІЧНОЇ МОВИ TURBO – PASCAL

2.1.Вступ до мови Pascal

2.1.1.Загальна характеристика алгоритмічних мов

Програма є набором команд на мові, зрозумілій виконавцю, що реалізовує деякий алгоритм. У нашому випадку виконавцем є комп'ютер, а мовою програмування буде мова високого рівня Pascal. На жаль, будь-яка мова високого рівня зручна тільки людині, що пише або відлажує програму, але абсолютно незрозуміла комп'ютеру. Програма на такій мові називається початковим текстом і зберігається в зовнішньому файлі з розширенням .txt.

Для перекладу програми на мову низького рівня, зрозумілу виконавцю-комп'ютеру, існують спеціальні програми-перекладачі – ***компілятори***. Результатом роботи компілятора (іншими словами, результатом процесу компіляції) є виконуваний код**,** який записується у файл з розширенням .exe.

Алгоритмічна мова високого рівня Pascal була розроблена в кінці 60-х років професором Н.Віртом. Вона була створена спеціально для навчання програмуванню. До основних достоїнств мови Pascal слід віднести гнучкість і надійність, простоту і ясність конструкцій, можливість задоволення вимог структурного програмування, наявність набору структурованих типів даних: масивів, записів, записів з варіантами, файлів, множин, можливість побудови нових типів даних.

На базі стандартної мови Pascal фірма Borland розробила сімейство Pascal - систем, званих Turbo – Pascal. Интегрована система Turbo – Pascal користується широкою популярністю серед масових користувачів і професійних програмістів. Це пояснюється наявністю дуже зручного інтегрованого середовища і тим, що в його основі лежить могутня мова програмування, що є розширеною версією мови Pascal.

За останні роки фірма Borland розробила і випустила на ринок сім модифікацій цієї системи. Кожна з них є удосконаленням попередньої. Безперервне вдосконалення системи Turbo – Pascal породило врешті-решт дуже могутню по своїх можливостях систему програмування, що відповідає найвищим вимогам. За допомогою Turbo – Pascal можна створювати багато програм — від програм, призначених для вирішення простих обчислювальних завдань, до складних сучасних систем управління базами даних і операційних систем.

І разом з тим Turbo – Pascal залишається простою у вивченні, що дозволяє програмісту, що починає, на її підставі вивчити методи і способи ефективного програмування.

2.1.2.Загальна структура програми на мові Turbo – Pascal

У складі середовища розробника Turbo Pascal є:

- текстовий редактор, в якому можна набирати тексти програм;
- компілятор, що перетворює початкові тексти на виконуваний код;
- відладчик, що допомагає виявляти і виправляти помилки в програмі.

Будь-який компілятор вимагає, щоб програма, що подається йому для перекладу, була абсолютно правильно складена.

У мові програмування, як і в будь-якій іншій мові, існують ***синтаксис*** – правила запису його конструкцій і ***семантика*** – сенс його конструкцій. Компілятор перевіряє тільки синтаксис. Пошуком же семантичних помилок займається програміст в процесі тестування і відладки своєї програми. Щоб зменшити витрати часу і сил на відладку, потрібно писати синтаксично і логічно правильні програми.

Програма на мові Turbo – Pascal складається із заголовка і власне програми, званої ***блоком***. Блок складається з розділів. Максимальна кількість розділів - шість. Розділи розташовуються в наступному порядку:

1. Розділ міток.
2. Розділ констант.

3. Розділ типів.
4. Розділ змінних.
5. Розділ процедур і функцій.
6. Розділ операторів.

Розділ операторів полягає в операторні дужки ***BEGIN ... END***. У ньому указується послідовність дій, які повинні виконуватися ЕОМ. Вся решта розділів носить описовий характер.

Будь-який розділ, окрім останнього, може бути відсутнім. Роздільником між розділами і операторами служить крапка з комою. В кінці програми повинна стояти крапка.

У будь-яке місце програми можуть бути включені коментарі. При цьому сенс програми не міняється. Коментарі включаються у фігурні дужки. Пізні версії компіляторів мови Pascal вже не вимагають указувати назву програми, тобто рядок **`program <ім'я_програми>;`** можна опустити. Але це можливо тільки в тому випадку, якщо вся програма міститься в одному модулі-файлі. Якщо ж програма складається з декількох самостійних шматків-модулів, то кожний з них повинен мати заголовок **`(program`** або **`unit)`**.

Будь-який з перерахованих необов'язкових розділів може зустрічатися в тексті програми більше одного разу, їх загальна послідовність також може мінятися, але при цьому завжди повинне виконуватися головне правило мови Pascal: перш ніж об'єкт буде використаний, він повинен бути оголошений і описаний.

Компілятори мови Pascal не розрізняють рядкові і прописні букви, а пробільні символи ігнорують, тому текст програми можна структурувати так, щоб читати і відлажувати його було найзручніше.

Великі логічно замкнуті блоки програми зручно розділяти рядками-коментарями, що містять інформацію про сенс подальшого блоку. ***Коментар*** – це рядок (або декілька рядків) з довільних символів, ув'язнених у фігурні дужки:

```
{ коментар }
```

Інший варіант оформлення коментаря:

```
(* коментар *)
```

Усередині самого коментаря символи } або *) зустрічатися не повинні.

Під час компіляції програми коментарі ігноруються. Отже, їх можна додавати в будь-якому місці програми. Можна навіть розірвати оператора вставкою коментаря. Крім того, все, що знаходиться після ключового слова ***end***, що завершує текст програми, компілятор теж сприймає як коментар.

2.1.3.Алфавіт мови Pascal

При написанні програм на алгоритмічній мові можна користуватися лише символами, передбаченими алфавітом цієї мови.

Алфавіт мови Turbo Pascal включає наступні символи.

Рядкові і прописні букви латинського алфавіту:

| a | b | c | .| x | y | z | A | B | C |.| X | Y | Z |.

Арабські цифри:

| 0 | 1 | 2 |..| 9 |.

Спеціальні символи:

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

Таблиця 2.1. – Перелік зарезервованих слів мови Pascal

and	**та**	**array**	**масив**
begin	**початок**	**case**	**випадок**
const	**константи**	**div**	**div**
do	**виконати**	**downto**	**вниздо**
else	**інакше**	**end**	**кінець**
for	**для**	**function**	**функція**
if	**якщо**	**mod**	**mod**
not	**не**	**of**	**із**
or	**або**	**procedure**	**процедура**
program	**програма**	**record**	**запис**
repeat	**повторювати**	**then**	**то**
to	**до**	**type**	**типи**
until	**покине**	**var**	**змінні**
while	**поки**	**label**	**мітки**
goto	**йтидо**	**string**	**рядки**

Крім того, в алфавіт Turbo Pascal включається набір зарезервованих слів, що мають строго певне призначення. Призначення окремих зарезервованих слів пояснюється в табл.2.1.

2.1.4.Константи і змінні

Константами є величини, які не міняють свого значення при виконанні програми. Як константи в Turbo – Pascal використовуються цілі і дійсні числа, логічні константи, символи і рядки.

Цілі числа записуються із знаком або без нього за звичайними арифметичними правилами.

Наприклад: 15 +1000 -47 02.

Дійсні числа можуть записуватися або у формі десяткового числа, або з вказівкою порядку.

У записі десяткового числа ціла частина відділяється від дробовою десятковою крапкою.

Наприклад: 2.5 -14.0 +0.33 0.0.

Числа з вказівкою порядку мають вигляд: $a \cdot 10^p$,

де a – є мантисою; p – десятковим порядком.

У Turbo – Pascal числа з порядком записуються з використанням букви E, за якою слідує порядок. Буква E читається як “помножити на 10 в ступені”.

Таблиця 2.2 – Приклади запису чисел з порядком

Математичний запис	**Запис в Turbo – Pascal**
$3.14 \cdot 10^5$	3.14E5
$-17 \cdot 10^{-2}$	-17E-2
25.625	2.5625E1
10^{-6}	1E-6
0.00048	4.8E-4

Мантиса може бути цілим або десятковим числом. ***Порядок*** завжди є цілим числом (табл. 2.2). Слід пам'ятати, що в написанні дійсного числа з порядком повинні бути обов'язково присутніми і мантиса, і порядок.

Логічні константи можуть приймати одне з двох логічних значень, записаних або словом TRUE (істина), або словом FALSE (брехня).

Символьна константа – це будь-який символ ПК, поміщений в апострофи, наприклад:

‘Y’ ‘Д’ ‘!’ ‘_’.

Строкова константа – будь-яка послідовність символів з набору символів ПК, ув’язнена в апострофи, наприклад:

‘X =’ ;

‘Максимальне значення =’.

Рядки використовуються при виведенні текстів і коментарів.

Змінними називаються величини, значення яких можуть змінюватися в процесі виконання програми. У Turbo – Pascal змінна задається ім'ям. Ім'я є послідовністю букв і цифр, що починається з букви. У імені може бути присутнім символ підкреслення. Довжина імені складає від 1 до 63 символів. Слід пам'ятати, що пропуски не повинні входити в написання імені.

Таблиця 2.3 – Приклади запису імен змінних

Математичний запис	x	ap	y^1	α	Σ	d-27
Запис в Турбо Паскалі	x	ap	y^1	alpha	S	d_27

Змінні, подібно до констант, можуть бути різних типів. ***Тип даних*** – це характеристика діапазону значень, які можуть приймати змінні, що відносяться до цього типу даних.

Типи задаються стандартними іменами.

INTEGER – цілий тип;

REAL – дійсний тип;

BOOLEAN – логічний тип;

CHAR – символьний тип;

STRING – строковий тип.

У програмі перед використанням змінної в спеціальному розділі описів **VAR** необхідно оголосити тип змінної. Однотипні змінні в розділі перераховуються через кому і відділяються від оголошеного типа двокрапкою. Після визначення типу ставиться крапка з комою.

Приклад розділу опису змінних:

```
var k, l:integer;
x,y,z: real;
t: boolean;
w: char;
stv: string;
```

По опису змінної в пам'яті комп'ютера резервується осередок для зберігання її значення. Залежно від оголошеного типа осередок може мати різну внутрішню структуру, тобто містити різне число байт.

2.1.5.Стандартні типи даних

Цілий тип забезпечує завдання цілих чисел. Існує декілька видів цілих типів: **byte, shortint, integer, longint.**

У таблиці 2.4 маємо: x – вираз будь-якого з типів;

b, l, i, w – вирази відповідних типів: byte, longint, integer, word;

vx – змінна типу x.

Арифметичні операції: +(складання), -(віднімання) *(множення),

/(ділення), DIV(ділення без остачі), MOD(обчислення залишку від цілочисельного ділення).

Операції відношення: =(рівно), <>(не рівно), <(менше), >(більше), <=(менше або рівно), >=(більше або рівно).

Над даними ***дійсного типа*** визначені арифметичні операції: +(складання), -(віднімання) *(множення), /(ділення), а також операції відношення (див. вище).

Таблиця 2.4 – Вбудовані процедури і функції, застосовні до цілих типів.

Звернення	Тип результату	Дія
abs(x)	x	Повертає модуль x
chr(b)	Char	Повертає символ за його кодом
dec(vx[,i])	процедура	Зменшує значення vx на i, за відсутності і на 1
inc(vx[,i])	-/-	Збільшує значення vx на i, за відсутності і на 1
odd(l)	boolean	Повертає TRUE, якщо аргумент непарне число, FALSE – якщо парне
random(w)	як у параметра	Повертає псевдовипадкове число, рівномірно розподілене на інтервалі 0<=x<w
sqr(x)	-/-	Повертає квадрат аргументу
exp(x)	real	e^x
sqrt(x)	real	Повертає квадратний корінь з x
sin(x)	-/-	sin x
cos(x)	-/-	cos x
ln(x)	-/-	ln x
arctan(x)	-/-	arctg x
succ(x)	як у параметра	Повертає наступне ціле число, тобто x+1
pred(x)	-/-	Повертає попереднє ціле число, тобто x-1

Значеннями ***символьного типа*** є елементи кінцевого і впорядкованого набору знаків. Символ, поміщений в апострофи, позначає константу символьного типа, наприклад: ‘5’, ‘d’.

Над змінними символьного типа визначені наступні функції:

1. функції перетворення:

ORD (s) – дає порядковий номер символу s у впорядкованій безлічі символів: **ORD(‘5’)=53.**

CHR (i) – дає символ, що стоїть під номером i у впорядкованій безлічі символів: **CHR(66)=‘B’.**

2. операції відношення (див. вище):

якщо C1 і C2 – символьні змінні, то C1>C2 істинно тільки тоді, коли **ORD(C1)>ORD(C2).**

3. стандартні функції:

PRED (s) – повертає попередній символ, **SUCC (s)** – повертає наступний символ.

Таблиця 2.5 – Вбудовані процедури і функції, вживані до дійсних типів

Звернення	**Тип параме-тра**	**Тип резуль-тату**	**Дія**
abs(x)	real, integer	x	Повертає модуль x
random(x)	integer	integer	Повертає псевдовипадкове число, рівномірно розподілене на інтервалі 0<=i<x
sqr(x)	real, integer	тип аргументу	Повертає квадрат аргументу
exp(x)	real	real	ex
sqrt(x)	real	real	Повертає квадратний корінь з x
sin(x)	-/-	-/-	sin x
cos(x)	-/-	-/-	cos x
ln(x)	-/-	-/-	ln x
arctg(x)	-/-	-/-	arctg x
frac(x)	-/-	-/-	Дробова частина числа
int(x)	-/-	-/-	Ціла частина числа
pi	---	real	Пі=3.14159265...
trunc(x)	real	integer	Відкидання дробової частини
round(x)	real	integer	Округлення до найближчого цілого
randomize	---	---	Ініціація датчика псевдовипадкових чисел
random	---	real	Повертає псевдовипадкове число, рівномірно розподілене на інтервалі 0<=x<1

Змінні ***булевого типа*** можуть приймати тільки два значення: **TRUE і FALSE.**

Над ними визначені:

1. логічні операції: **AND** (і або кон'юнкція), **OR** (або або диз'юнкція), **NOT** (не або заперечення).

2. операції відношення (див. вище): причому **TRUE>FALSE**.

Перелічуваний тип задається переліком тих значень, які може одержувати змінна цього типа. Нумерація в списку значень починається з 0, тобто перше значення має номер – 0, а друге – 1 і т.д.

Приклад: **type tree=(birch, oak, pine);**

Або **var tr=(birch, oak, pine);**

2.1.6. Вирази

Значення виразів обчислюються з урахуванням розставлених дужок і старшинства операцій. Нижче приведені операції у порядку убування їх пріоритету, причому операції в одному рядку мають однаковий пріоритет:

```
NOT
*, /, MOD, DIV, AND
+, -, OR
<, >, <=, >=, <>, =
```

Операції одного і того ж старшинства виконуються зліва направо у порядку їх появи у виразі. Вирази в круглих дужках обчислюються в першу чергу.

Арифметичні вирази будуються з операндів, арифметичних операцій і круглих дужок.

Як операнди можуть виступати константи, змінні і функції.

У бездужкових арифметичних виразах операції виконуються зліва направо відповідно до їх пріоритету.

1. ***** (множення);
   / (ділення);
   DIV (ділення без остачі);
   MOD (виділення залишку від ділення цілих чисел).
2. **+** (складання);
   – (віднімання).

Змінити порядок виконання операцій можна за допомогою круглих дужок. Вираз, поміщений в круглі дужки, виконується в першу чергу. Наприклад, виразу a/bc відповідає математичний запис $\frac{a}{b}c$, а виразу a/(bc) відповідає запис $\frac{a}{bc}$.

Тип арифметичного виразу визначається типом операндів, що входять в нього. Арифметичний вираз є цілим, якщо всі складаючі його операнди є цілого типу. Результат обчислення цілого арифметичного виразу – цілий. Якщо в арифметичному виразі міститься хоч би один дійсний операнд, то результат – дійсний. Цілі операнди в речовинному арифметичному виразі завжди перетворяться до дійсного типу. Операції цілочисельного ділення **DIV** і **MOD** застосовні тільки до цілих чисел. Результат їх виконання – цілий. Виняток становить операція ділення з використанням символу '/'. Результат виконання цієї операції завжди дійсний, незалежно від типу операндів. Наприклад, значенням виразу 2/5 буде дійсне число 0.4 (табл..2.6).

Таблиця 2.6 – Приклади обчислень арифметичних виразів

Арифметичні вирази	**Результат**	**Тип результату**
6 + 4 * (5 – 3)	14	Цілий
6 + 4 * (5 – 3.0)	14.0	Дійсний
7 DIV 2	3	Цілий
7 MOD 2	1	Цілий
7/2	3.5	Дійсний

Часто використовувані в арифметичних виразах елементарні математичні функції оформлені у вигляді стандартних підпрограм, які зберігаються в бібліотеці Turbo – Pascal (файл з ім'ям TURBO.TPL).

Для правильного звернення до стандартної функції, необхідно записати ім'я функції, за яким в круглих дужках слідує аргумент (параметр). Пріоритет обчислення функцій вищий, ніж пріоритет арифметичних операцій.

Нижче, в таблиці 2.7 представлений набір стандартних функцій з вказівкою типів функції і аргументу, в таблиці прийняті позначення: I-цілий тип, R-дійсний тип.

У Turbo – Pascal відсутня операція зведення в ступінь. Реалізація цієї операції залежить від типу показника. Якщо показник цілого типа, то операція зведення в ступінь реалізується багатократним множенням.

Наприклад:

```
x⁴ =>x * x * x * x;
sin²x => sqr(sin(x)).
```

Таблиця 2.7 –Стандартні математичні функції в Turbo – Pascal

Математичний запис	**Запис в Turbo – Pascal**	**Тип аргументу**	**Тип функції**
|x|	**Abs(x)**	I R	I R
x^2	**Sqr(x)**	I R	I R
sin x **cos x** **arctg x**	**Sin(x)** **Cos(x)** **Arctan(x)**	I або R	R
ex **ln x**	**Exp(x)** **Ln(x)**	I або R	R
$\sqrt{x}$	**Sqrt(x)**	I або R	R
π	**Pi**	R	R
Виділення цілої частини аргументу	Trunc(x)	R	I
Виділення дробової частини аргументу	Frac(x)	R	R

Якщо показник дійсного типа, то користуються співвідношенням: $x^a = e^{a \ln x}$,

де x>0; a – дійсне число або вираз.

Запис в Turbo – Pascal має вигляд: x^a=>exp(a*ln(x)).

Наприклад: $\sqrt[3]{x} = x^{\frac{1}{3}} \Rightarrow \exp(1/3 * \ln(x))$.

Таблиця 2.8 – Приклади програмування арифметичних виразів

Математичний запис	**Запис в Turbo – Pascal**
$\frac{a+12b}{c_1 - 1.8 \cdot 10^3}$	(a + 12 * b)/(c1 - 1.8E3)
$e^{\frac{2\sin 4x + \cos^2(x^2)}{3x}}$	exp((2sin(4x)+sqr(cos(xx)))/(3*x))
ln |tgα-sin(3)|	ln(abs(sin(al) /cos(al) -sin(al*al*al)))

2.2.Оператори мови Pascal

Перейдемо тепер до вивчення ***операторів*** – спеціальних конструкцій мови Pascal. Якщо говорити строго, то ***оператором*** називається (мінімальна) структурно закінчена одиниця програми.

Всі оператори мови Pascal повинні закінчуватися знаком **";"** (крапка з комою), і жоден оператор не може розриватися цим знаком. Єдина можливість не ставити після оператора ";" з'являється у тому випадку, коли відразу за цим оператором слідує ключове слово ***end***.

2.2.1.Прості оператори мови Pascal

До простих операторів мови Pascal відносяться:

1. a:= b; – ***привласнення*** змінної a значення змінної b. У правій частині привласнення може знаходитися змінна, константа, арифметичний вираз або виклик функції.

2. **;** – ***порожній*** оператор, який можна вставляти куди завгодно, а також викреслювати звідки завгодно, оскільки на цілісність програми це ніяк не впливає.

3. ***операторні дужки***, що перетворюють декілька операторів в один:

```
begin
<несколько операторов>
end;
```

Скрізь далі, де в записі конструкцій мови Pascal ми використовуватимемо позначення <один_оператор>, його слід розуміти як "один оператор або декілька операторів, ув'язнених в операторні дужки **begin - end**".

2.2.2.Мітки і безумовний перехід

Мітка позначає яке-небудь місце в тексті програми. Мітками можуть бути числа від 0 до 9999 або ідентифікатори, які в цьому випадку вже не можна використовувати для яких-небудь інших потреб. Всі мітки повинні бути описані в спеціальному розділі label таким чином:

```
label <список_всіх_міток_через_кому>;
```

Міткою може бути помічений будь-який оператор програми

```
<мітка>: <оператор>;
```

Будь-яка мітка може зустрітися в тексті програми тільки один раз. Використовуються мітки тільки ***операторами безумовного переходу*** `GOTO`***:***

```
goto <метка>;
```

Це означає, що відразу після оператора `GOTO` буде виконаний не наступний за ним оператор (як це відбувається в звичайному випадку), а той оператор, який помічений відповідною міткою.

Передавати управління можна вперед і назад по тексту програми, всередину складених операторів і назовні і т.п. Виключенням є тільки процедури і функції (див. далі): всередину їх і назовні безумовні переходи неможливі.

2.2.3.Операції введення

Будь-який алгоритм повинен бути результативним. У загальному випадку це означає, що він повинен повідомляти результат своєї роботи споживачу: користувачу-людині або іншій програмі (наприклад, програмі управління принтером). Розглянемо докладніше процеси введення інформації з клавіатури і

виведення її на екран. У програмуванні існує спеціальне поняття ***консоль***, яке позначає клавіатуру при введенні і монітор при виведення.

Для того, щоб одержати дані, що вводяться користувачем уручну (тобто з консолі), застосовуються команди

read(<список_ввода>) і **readln(<список_ввода>).**

Перша з цих команд прочитує всі запропоновані їй дані, залишаючи курсор в кінці останнього рядка введення, а друга – відразу після закінчення введення переводить курсор на початок наступного рядка. У іншому ж їх дії повністю співпадають.

Список введення – це послідовність імен змінних, розділених комами. Наприклад, за допомогою команди

```
readln(k,x,c,s);      {k:byte; x:real; c:char; s:string}
```

програма може одержати з клавіатури дані відразу для чотирьох змінних, що відносяться до різних типів.

Значення, що вводяться, необхідно розділяти пропусками, а завершувати введення – натисненням клавіші ***Enter***. Введення даних закінчується в той момент, коли остання змінна із списку введення набула свого значення. Отже, вводячи дані за допомогою приведеної вище команди, ви можете натиснути ***Enter*** чотири рази – після кожної із змінних, що вводяться, або ж тільки один раз, заздалегідь ввівши всі чотири змінні в одну строчку (обов'язково потрібно розділити їх пропусками). Типи значень, що вводяться, повинні співпадати з типами вказаних змінних, інакше виникає помилка. Тому потрібно уважно стежити за правильністю даних, що вводяться.

Взагалі, вводити з клавіатури можна тільки дані базових типів (за винятком логічного). Якщо ж програмі все-таки необхідно одержати з консолі значення для boolean – величини, доведеться діяти хитріше: вводити обумовлений символ, а вже на його основі привласнювати логічній змінній відповідне значення.

Наприклад:

```
repeat
writeln('Чи згодні Ви з цим твердженням? y – так, n – ні');
readln(c);     c:char}
```

```
case c of
'y': b:= true;
'n': b:= false;
else: writeln('Помилка!');
until (c='n') or(c='y');
```

Друге виключення: рядки, хоча вони і не є базовим типом, вводити теж дозволяється. Ознакою закінчення введення рядка є натиснення клавіші ***Enter***, тому всі наступні за нею змінні необхідно вводити з нової строчки.

2.2.4.Операції виведення

Для того, щоб вивести на екран яке-небудь повідомлення, користуються процедурою

write(<*список_виводу*>) або writeln(<список_виводу>).

Перша з них, надрукувавши на екрані все, про що її просили, залишить курсор в кінці виведеного рядка, а друга переведе його в початок наступної строчки.

Список виведення може складатися з декількох змінних, записаних через кому; всі ці змінні повинні мати тип або базовий, або рядковий. Наприклад,

```
writeln(a,b,c);
```

Якщо для виведення інформації скористатися командою **writeln(a,b,c)**, то символи, що виводяться, виявляться "зліпленими". Щоб цього не трапилося, потрібне або поклопотатися про пропуски між змінними, що виводяться:

```
writeln(a,' ',b,' ',c);
```

або задати для деяких змінних (а і з) формат виведення*:*

```
writeln(a:5,b,c:20:5);
```

У ***форматному виведенні*** перше число після знаку **":"** позначає кількість позицій, що виділяються під всю змінну, а друге - під дробову частину числа. Десяткова крапка теж вважається окремим символом. Якщо число довше, ніж відведений під нього простір, кількість позицій буде автоматично збільшено. Якщо ж число коротше заданого формату, що виводиться, то спереду до нього припишуться декілька пропусків. Таким чином, можна проводити висновок красивими рівними стовпчиками, а також стежити за тим, щоб змінні не зливалися.

Наприклад, якщо a = 25, b = 'x', а c = 10.5, то після виконання команди **`writeln(a:5,' ',b,c:10:5)`** на екрані або у файлі буде записано наступне (підкреслення в даному випадку служать лише для візуалізації пропусків):

_ _ _25_x_ _10.50000

Особливо важливий формат при виведенні речовинних змінних. Наприклад, якщо не вказати формат, то число 10.5 буде виведене як 1.0500000000E+0001. Такий формат називається записом з ***плаваючою крапкою***.

Таблиця 2.9 – Приклади виведення, формату дійсних чисел

Оператор форматного виведення	Результат виведення на екран
`write (125.2367:10);`	`_1.3E+0002`
`write (125.2367:11);`	`_1.25E+0002`
`write (125.2367:12);`	`_1.252E+0002`
`write (125.2367:13);`	`_1.2524E+0002`
`write (125.2367:14);`	`_1.25237E+0002`
`write (125.2367:15);`	`_1.252367E+0002`
`write (125.2367:16);`	`_1.2523670E+0002`

Якщо ж задати тільки загальну довжину дійсного числа, не указуючи довжину дробової частини, то воно займатиме на екрані задану кількість символів (у разі потреби, спереду буде додана відповідна кількість пропусків), але при цьому залишиться у форматі плаваючої крапки. Мінімальною довжиною для виведення дійсних чисел є 10 (при форматі _x.xE+yyyy). Перша позиція зарезервована під знак "-". Необхідно пам'ятати, що у разі недостатньої довжини виведення число буде автоматично закруглено, наприклад (підкреслення служить для візуалізації пропуску).

Приклад найпростішої програми на мові Pascal

```
program start;
var s: string;
begin
```

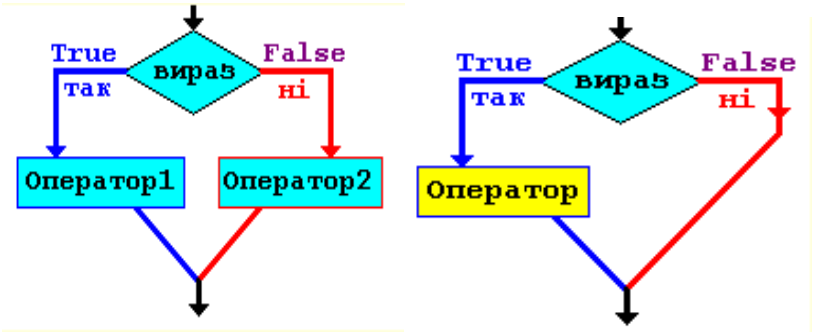
```
write('Будь ласка, введіть Ваше ім'я: ');
readln(s);
writeln('Ми раді Вас вітати, ',s,'!'); end.
```

2.2.5.Умовний оператор *IF*

Оператор ***IF*** вибирає між двома варіантами розвитку подій:

```
if <условие>
then <один_оператор>
[else <один_оператор>];
```

Зверніть увагу, що квадратні дужки вигляду [ ] показують, що виділена ними частина синтаксису є необов'язковою. Перед словом **else** (коли воно присутнє) символ ";" не ставиться – адже це розірвало б оператора на дві частини.



а) б)

Рисунок 2.1 – Блок-схема повного (а) та неповного (б) умовного оператора

Умовний оператор ***IF*** працює таким чином:

1. Спочатку обчислюється значення <условия> – це може бути будь-який вираз, що повертає значення типу **boolean**.

2. Потім, якщо в результаті одержана "істина" (**true**), то виконується оператор, що стоїть після ключового слова **then**, а якщо "брехня" (**false**) – без додаткових перевірок виконується оператор, що стоїть після ключового слова **else**. Якщо ж else–гілка відсутня, то не виконується нічого.

Що ж відбудеться, якщо написати декілька вкладених операторів ***IF*** ? У разі, коли кожен оператор ***IF*** має власну ***else*** – гілку, все буде в порядку. А ось якщо деякі з них цієї гілки не мають, може виникнути помилка. Компілятор

мови Pascal завжди вважає, що ***else*** відноситься до самого ближнього оператора ***IF***. Таким чином, якщо написати

```
if i>0 then if s>2
then s:= 1
else s:= -1;
```

маючи на увазі, що ***else***–гілка відноситься до зовнішнього оператора ***if***, то компілятор все одно сприйме цей запис як

```
if i>0 then if s>2
then s:= 1
else s:= -1
else
```

Ясно, що таким чином правильного результату отримати не вдасться.

Для того, щоб уникнути подібних помилок, варто завжди (або принаймні за наявності декількох вкладених умовних операторів) указувати обидва ключові слова, навіть якщо одна з гілок буде порожньою.

Отже, початковий варіант потрібно переписати таким чином:

```
if i>0 then if s>2
then s:=1
else
else s:=-1;
```

або так:

```
if i>0 then begin if s>2
then s:=1
end
else s:=-1;
```

Взагалі ж, якщо є можливість переписати декілька вкладених умовних операторів як один оператор ***вибору***, це варто зробити.

2.2.6.Оператор вибору *CASE*

Оператор ***CASE*** дозволяє зробити вибір між декількома варіантами:

```
case <перемикач> of
     <список_констант> : <один_оператор>;
     [<список_констант> : <один_оператор>;]
     [<список_констант> : <один_оператор>;]
     else <один_оператор>;]
     end;
```

Зверніть увагу, що після ***else*** двокрапка не ставиться.

Існують додаткові правила, що відносяться до структури цього оператора:

1. Перемикач повинен відноситися тільки до порядкового типа даних, але не до типу longint.
2. Перемикач може бути змінною або виразом.
3. Список констант може задаватися як явним переліком, так і інтервалом або їх об'єднанням.
4. Повторення констант не допускається.
5. Тип перемикача і типи всіх констант повинні бути сумісними.

Приклад оператора вибору:

```
case symbol(* :char *) of
'a'..'z', 'A..'Z' : writeln('Це латинська буква');
'a'..'я', 'A..'Я' : writeln('Це російська буква');
'0'..'9' :          writeln('Це цифра');
' ',#10,#13,#26     :   writeln('Це пробільний символ');
else              writeln('Це службовий символ');
end;
```

Виконання оператора ***CASE*** відбувається таким чином:

1. обчислюється значення перемикача;
2. отриманий результат перевіряється на приналежність до того або іншого списку констант;
3. якщо такий список знайдений, то подальші перевірки вже не проводяться, а виконується оператор, відповідний вибраній гілці, після чого управління передається оператору, наступному за ключовим словом ***end***, яке закриває всю конструкцію ***case***.
4. якщо відповідного списку констант немає, то виконується оператор, що стоїть за ключовим словом ***else***. Якщо ***else***–гілки немає, то не виконується нічого.

2.2.7. Оператор циклу з передумовою

Оператор циклу з передумовою містить у собі логічний вираз, який керує повторним виконанням оператора (що може бути складеним оператором).

```
While вираз do
```

Оператор

Поки вираз ***TRUE*** треба виконувати оператор

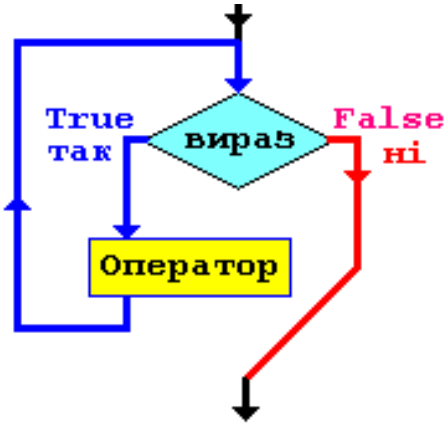


Рисунок 2.2. – Блок -схема циклу з передумовою

Вираз, за допомогою якого здійснюється керування повторенням оператора, повинний мати логічний тип. Обчислення його проводиться до того, як внутрішній оператор буде виконаний. Внутрішній оператор виконується повторно доти, поки вираз приймає значення ***false***. якщо вираз із самого початку приймає значення ***false***, то оператор, що міститься всередині оператора циклу з передумовою, не виконується.

Приклад оператора циклу з передумовою:

```
while Data[ A ]< > X do
A := A + 15.
```

3.1.1. Оператор циклу з післяумовою

Оператор циклу з післяумовою записують в загальному вигляді так:

```
Repeat оператор1;
........
операторN
until вираз
```

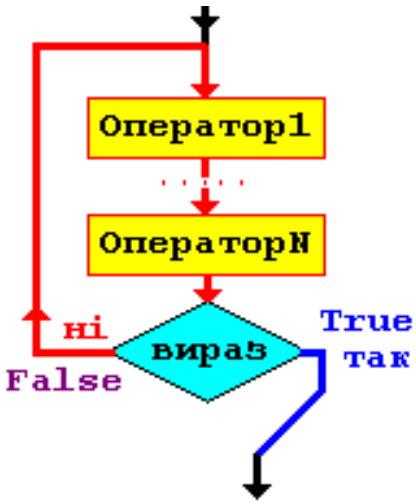


Рисунок 2.3. – Блок-схема циклу з післяумовою

Операторів може бути довільна кількість, або не бути зовсім. Результат виразу повинний мати логічний тип. Оператори, поміщені між ключовими словами ***repeat*** і ***until***, виконуються послідовно доти, поки результат виразу не прийме значення ***true***. Послідовність операторів виконується принаймні один раз, оскільки обчислення виразу проводиться після виконання послідовності операторів.

Приведемо приклад оператора циклу з післяумовою:

```
Repeat
K := I mod J;
I := J;
J := K;
until J = 0;
```

2.2.8. Оператор циклу з параметром

Оператор циклу з параметром викликає повторюване виконання оператору (що може бути складеним оператором) для всіх значень керуючої змінної в межах заданого діапазону.

```
For змінна := ПочЗн to КінЗн do оператор
```

У якості керуючої змінної повинен використовуватися ідентифікатор, що позначає змінну, оголошену локальною в блоці, у якому міститься оператор FOR. Керуюча змінна повинна мати простий порядковий тип. Початкове і кінцеве значення повинні мати тип, сумісний по присвоюванню з порядковим типом. Коли починає виконуватися оператор FOR, початкове і кінцеве значення визначаються

один раз і ці значення зберігаються протягом усього виконання оператора. Оператор, що міститься у тілі оператора FOR, виконується один раз для кожного значення в діапазоні між початковим і кінцевим значенням. Керуюча змінна завжди ініціалізується початковим значенням.

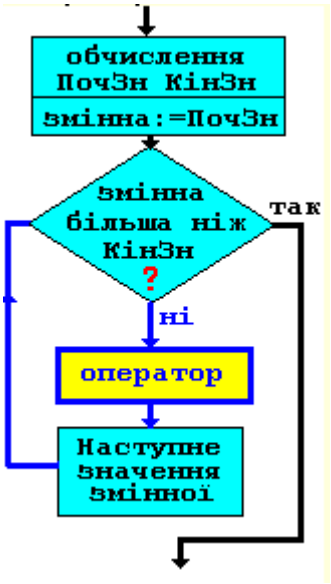


Рисунок 2.4 – Блок-схема циклу з параметром

При кожному наступному повторенні оператора ***FOR*** береться наступне значення керуючої змінної. Якщо початкове значення перевищує кінцеве значення, то оператор, що міститься у тілі оператора ***FOR***, не виконується. Колі в операторі циклу замість слова ***do*** використовується ключове слово ***downto*** (внизДо *)*, значення керуючої змінної змінюються в зворотньому порядку. якщо початкове значення в цьому випадку менше, ніж кінцеве значення, то оператор не виконується. Оператор, що міститься у тілі оператора ***FOR***, не повинен змінювати значення керуючої змінної. Після виконання оператора ***FOR*** значення керуючої змінної стає невизначеним, якщо тільки виконання оператора ***FOR*** не було перервано за допомогою оператора переходу.

Приведемо приклад використання оператора циклу з параметром:

```
For sym:=z downto про do
If str[i]= sym then GoTo 25;
For I := 1 to 10 do
For J := 1 to 10 do
```

```
Begin
X:= 0;
For K:= 1 to 10 do
X:= X + Mat1[I,K]*Mat2[K,J];
Mat[I,J]:=X;
end;
```

2.3. Основи практичної роботи в середовищі Turbo Pascal

Інтегроване середовище об'єднує текстовий редактор, компілятор, відладчик і довідкову систему. Всі ці складові частини необхідні для успішної роботи по створенню початкового тексту програми, його компіляції, запуску і пошуку можливих помилок на етапі виконання. Не дивлячись на те, що інтегроване середовище Turbo Pascal створювалося достатньо давно (за мірками світу комп'ютерних технологій, що стрімко розвивається), деякі її риси збереглися і в сучасних складних системах розробки програм.

Процес створення програми на ЕОМ включає декілька етапів.

1. Набір початкового тексту програми (текстовий редактор).
2. Компіляція і усунення синтаксичних помилок (компілятор, текстовий редактор).
3. Виконання програми і її відладка (виконуюча система, відладчик). Інтегроване середовище Turbo Pascal містить всі необхідні для розробки програми засоби.

Для того, щоб працювати в інтегрованому середовищі Turbo Pascal, необхідно встановити на комп'ютері весь програмний пакет. Докладний опис процесу установки міститься в керівництві користувача, що додається до дистрибутивних дисків. Треба мати на увазі, що даний пакет призначений для роботи в операційній системі MS-DOS, з цим пов'язані певні обмеження і проблеми, з якими може зіткнутися користувач. По-друге, для установки пакету слід відвести певне місце на жорсткому диску — 20 Мбайт цілком достатньо.

Набір і редагування початкового тексту програми проводиться засобами вбудованого текстового редактора інтегрованого середовища. Якщо розкрити

меню Edit, то можна побачити перелік команд редактора. Цей перелік включає (по порядку) команди відміни попередньої дії (***Undo***), повторного виконання раніше скасованої дії (***Redo***), видалення в буфер обміну (***Cut***), копіювання в буфер (***Copy***), вставки з буфера (***Paste***) і «безповоротнього» видалення (***Clear***). Праворуч від імені команд показані пов'язані з ними клавіші або комбінації клавіш. Таке зіставлення використовується і в інших випадках.

Однією з найбільш важливих тут є команда ***Undo***. Для кожного редагованого файлу інтегроване середовище зберігає довгий запис зроблених в ньому змін і багатократним натисненням клавіш ***F10***, потім E, потім U (або ***Alt+Backspace***) можна відмінити внесені в програму зміни, якщо вони виявилися помилковими. Найбільш характерне застосування цієї команди — відновлення випадково стертого фрагменту (блоку) тексту.

Остання команда називається ***Show clipboard***. Вона дозволяє проглянути вміст буфера обміну. Річ у тому, що більшість текстових редакторів використовують спеціальний файл для часового зберігання переміщуваних або копійованих фрагментів тексту. При активізації команди ***Show clipboard*** відкривається нове вікно ***Clipboard***, в якому відображаються скопійовані в буфер обміну фрагменти тексту.

Основні можливості редактора зв'язані з використанням клавіатури і спеціальних комбінацій клавіш. Перелік спеціальних клавіш і комбінацій клавіш даний в таблицях 2.10 и 2.11.

Таблиця 2.10.Спеціальні клавіші і комбінації клавіш для редагування

Клавіші	**Команда меню**	**Функція**
Ctrl+Del	Edit > Clear	Видалення вибраного тексту
Ctrl+lns	Edit > Сміттю	Копіювання вибраного тексту в буфер
Shift+Del	Edit > Cut	Переміщення вибраного тексту в буфер
Shift+Ins	Edit > Paste	Запис тексту з буфера в активне вікно

Ctrl+L	Search > Search Again	Повтор останньої команди Find або Replace
F2	File > Save	Збереження файлу, що знаходиться в активному вікні редактора
F3	File > Open	Відкриття файлу

Таблиця 2.11 – Спеціальні комбінації клавіш редагування, не пов'язані з командами меню

Клавіші	**Функція**
Ctrl+K, потім R	Вставити в поточній позиції курсора файл з диска
Ctrl+K, потім W	Записати виділений блок у файл
Ctrl+K, потім будь-яка цифра	Встановити маркер з номером, відповідним цифрі
Ctrl+K, потім T	Виділити слово
Ctrl+Q, потім B	Перейти в початок виділеного блоку
Ctrl+Q, потім До	Перейти в кінець виділеного блоку
Ctrl+Q, потім будь-яка цифра	Перейти до маркера з номером, соответ. цифрі
Gtrl+Q, потім A	Знайти і замінити
Ctrl+Q, потім P	Перейти до попередньої позиції курсора
Ctrl+Q, потім L	Відновити рядок
Ctrl+Q, потім Y	Видалити текст до кінця рядка
Ctrl+Q, потім W	Відновити останнє повідомлення про помилку
Ctrl+Q, потім [	Знайти парну дужку в напрямі вперед
Ctrl+Q, потім]	Знайти парну дужку в напрямі назад
Ctrl+Y	Видалити рядок
Ctrl+N	Вставити рядок
Ctrl+Home	Перейти в початок екрану
Ctrl+End	Перейти в кінець екрану
Ctrl+PgUp	Перейти до початку файлу
Ctrl+PgDn	Перейти до кінця файлу
Shift+клавиша управління	Виділення блоку курсором

Створення нової програми відбувається таким чином. Після запуску інтегрованого середовища на екрані повинне з'явитися порожнє активне вікно редагування. Якщо вікно, що з'явилося, не порожнє, то за допомогою команди ***File > New*** слід відкрити нове вікно для введення тексту. У верхній частині вікна редагування з'явиться назва, яка середовище автоматично привласнює новому файлу — ***NONAMEOO.PAS.*** Після набору тексту програми треба обов'язково змінити ім'я файлу, інакше є небезпека втратити його, якщо він випадково буде заміщений іншим файлом з таким же стандартним ім'ям. Рекомендується також періодично проводити збереження файлу, натискаючи клавішу ***F2*** (через кожні 10-20 набраних рядків), оскільки завжди є вірогідність аварійної ситуації в роботі комп'ютера, після якої не збережений на диску файл буде втрачений. При першому записі файлу на диск система запропонує задати ім'я файлу, причому розширення ***.PAS*** додається автоматично.

З редагуванням пов'язано ще одне меню — ***Search***, команди якого використовуються для пошуку і заміни фрагментів тексту. Для того, щоб виконати пошук, необхідно вказати шуканий фрагмент тексту («зразок» для пошуку). Якщо курсор знаходиться в якомусь слові у вікні редагування, це слово за умовчанням буде зразком для пошуку у вікнах, що відкриваються командами ***Find*** (пошук) і ***Replace*** (заміна). Замість даного слова можна ввести будь-яке інше. У тому випадку, коли треба лише трохи змінити текст, за умовчанням виведений у вікно, необхідно перемістити курсор в полі вікна діалогу. Колір тексту поміняється, і зразок для пошуку можна буде відредагувати. Пошук може проводитися в різних режимах. Необхідний режим можна вибрати в діалоговому вікні.

Розглянемо команди меню ***File***, розташованого першим в рядку меню. Доступ до меню ***File*** можна одержати натисненням клавіші ***F10***, а потім F або клацанням на ньому лівої кнопки миші. Найчастіше використовуються команди ***Open, Save, Exit і DOS Shell*** цього меню. Першим трьом відповідають клавіші ***F3, F2*** і ***Alt+X***, тому для їх вибору можна і не розкривати меню ***File***. Команда закриття файлу ***Close*** з деяких причин розташована в меню ***Windows***, а не ***File***.

Файл можна закрити натисненням клавіш ***Alt+F3*** або клацанням на маленькому прямокутному значку в лівому верхньому кутку активного вікна. Якщо з моменту попереднього запису на диск виконувалося редагування файлу, послідує запрошення зберегти результати цього редагування.

Вибір команди ***File > Open*** веде до появи запрошення ввести ім'я файлу. Розширення .PAS додається до імені файлу автоматично. Якщо потрібно відкрити файл з розширенням, відмінним .PAS, слід ввести ім'я в одному з наступних форматів: NAME., або NAME.*, або *.* і т.д. залежно від того, який файл необхідно відкрити. Ім'я *.BAT, наприклад, дасть можливість роботи зі всіма файлами, відповідними цій масці. У нижній частині меню ***File*** знаходиться список останніх файлів, що відкривалися. На будь-якому з рядків цього списку можна клацнути мишею, щоб відкрити відповідний файл знов. Крім того, при виборі команди ***Open*** (або натисненні клавіші F3**)** також стає доступним список останніх файлів, що відкривалися.

У тому випадку, коли в інтегроване середовище завантажено декілька файлів, а це можливо при роботі над достатньо складною програмою, що складається з декількох частин, розташованих в різних файлах, важливим виявляється навик роботи з вікнами. Основні команди для роботи з вікнами винесені в спеціальне меню ***Window***. Меню ***Window*** містить команди ***Next*** і ***Previous*** (циклічне перемикання між вікнами), яким зіставлені клавіші ***F6*** і ***Shift+F6***. Там же є команда ***Close*** (закрити активне вікно). Можна змінювати розмір вікон, переміщати їх по робочому простору інтегрованого середовища (для цього служать команда ***Size/Move*** і комбінація клавіш ***Ctrl+F5***) і виконувати деякі інші дії. Після натиснення комбінації клавіш ***Ctrl+F5*** переміщення вікна проводиться за допомогою клавіш управління курсором, а для того, щоб змінити розмір активного вікна, при натисненні цих клавіш необхідно утримувати клавішу ***Shift***. Список відкритих вікон виводиться по команді ***List***. Перелік спеціальних комбінацій клавіш для управління вікнами приведений табл. 2.12.

Таблиця 2.12 – Спеціальні клавіші для управління вікнами

Клавіші	Команда меню	Функція
Alt+цифра 1-9		Перехід до вікна із заданим номером
Alt+O	Window > List	Показати список відкритих вікон
Alt+F3	Window > Close	Закрити активне вікно
Alt+F5	Window > User Screen	Показати екран користувача
Shift+F6	Window > Previous	Перехід до предыдущ. відкритому вікну
Ctrl+F5	Window > Size/Move	Зміна розміру (Shift+клавиши із стрілками) або положення активного вікна
F5	Window > Zoom	Розгортання активного вікна до розміру повного екрану або повернення вікна до початкового розміру
F6	Window > Next	Перехід до наступного відкритого вікна

В процесі набору тексту програми достатньо часто доводиться користуватися вбудованою довідковою системою Turbo Pascal. Довідкова система є контекстно-залежною, тобто інформація, що виводиться при зверненні до неї, залежить від того, де у момент звернення знаходився курсор.

Таблиця 2.13 – Спеціальні клавіші і комбінації клавіш вбудованої довідкової системи

Клавіші	Команда меню	Функція
F1	Help > Contents	Відкриває вікно довідкової системи
F1, F1	Help > Help on Help	Виклик довідки по довідковій системі
Shift+F1	Help > Index	Виклик змісту довідкової системи
Alt+F1	Help > Previous Topic	Показати попередній екран довідкової інформації
Ctrl+F1	Help > Topic Search	Виклик контекстної інформації по мові Pascal

При натисненні клавіші ***F1*** виводиться довідкова інформація, що відноситься до редагованого файлу або активізованого пункту меню. Найшвидший спосіб виклику довідкової системи — використання спеціальних клавіш або їх комбінацій (табл. 2.13).

Для того, щоб вийти з інтегрованого середовища, натисніть клавіші ***Alt+X***. При цьому буде запропоновано зберегти кожний з відкритих файлів, якщо результат їх редагування не був заздалегідь записаний на диск. На закінчення – декілька простих рад. Використання клавіші ***Esc*** часто допомагає вийти з важких ситуацій. Не бійтеся натискати клавішу ***Esc***, якщо при роботі в інтегрованому середовищі виникли проблеми! Клацання правою кнопкою миші виводить у вікно редагування контекстне меню, що містить невеликий, але актуальний для даного випадку набір команд. Це ж меню викликається натисненням клавіш ***Alt+F1O***.

Після завершення набору програми і її запису на диск можна приступати до компіляції (натиснувши клавішу ***F9***). При цьому компілятор може виявити помилки в програмі, і ці помилки треба буде виправити. Повідомлення про помилку виводиться у верхній частині екрану і виділяється червоним кольором. Курсор при цьому встановлюється в тому рядку програми, де виявлена помилка.

Виправлення помилок зводиться до внесення змін в початковий текст програми. При неясних ситуаціях, що відносяться як до характеру помилки, так і до правил мови або правил використання бібліотечних підпрограм, сміливо користуйтеся вбудованою довідковою системою Turbo Pascal.

Компіляція програми з інтегрованого середовища може проводитися в різних режимах. У меню ***Compile*** є команда ***Build***, не пов'язана з «гарячою» клавішею. Вона дозволяє перекомпілювати всі модулі, використовувані програмою, перш ніж почнеться компіляція самої програми. Команда ***Make*** (пов'язана з клавішею ***F9***) приводить до перекомпіляції тільки модифікованих модулів. Під час компіляції на екрані відображається вікно, що ілюструє хід компіляції. При

успішному завершенні компіляції в цьому вікні з'являється відповідне повідомлення.

Результат успішної компіляції — файл, ім'я якого співпадає з ім'ям початкового файлу, а розширення є стандартним розширенням виконуваного файлу MS-DOS, тобто. EXE. Цей файл може розміщуватися в оперативній пам'яті (в цьому випадку він буде втрачений при виході з інтегрованого середовища) або на диску.

Далі в меню ***Compile*** слідують команди, що дозволяють задати основний файл проекту, а також команда ***Information***, активізація якої дозволяє одержати корисну інформацію про скомпільовану програму (розмір програми, розподіл оперативної пам'яті і т. д.).

Таблиця 2.14 – Спеціальні клавіші для запуску і відладки програм

Клавіші	**Команда меню**	**Функція**
Alt+F9	Compile > Compile	Компіляція активного файлу
Ctrl+F2	Run > Program Reset	Повернення виконуваної програми в початковий стан
Ctrl+F4	Debug > Evaluate/Modify	Обчислення виразу або модифікація значення змінної
Ctrl+F7	Debug > Add Watch	Додавання виразу для перегляду
Ctrl+F8	Debug > Toggle Breakpoint	Установка або видалення точки останову
Ctrl+F9	Run > Run	Запуск програми на виконання
F4	Run > Go to Cursor	Виконання фрагмента програми від рядка, що підсвічується, до рядка, на якому знаходиться курсор
F7	Run > Trace Into	Запуск програми в режимі відладки із заходом всередину процедур
F8	Run > Step Over	Запуск програми в режимі відладки, минувши виклики процедур
F9	Compile > Make	Компіляція програми з поточного вікна

Завершивши «боротьбу» з синтаксичними помилками, можна запустити програму (натиснувши клавіші ***Ctrl+F9***). Програма може благополучно

відпрацювати, але можуть виявитися помилки часу виконання або алгоритмічні помилки. Якщо в програмі не передбачене припинення виконання для проглядання результатів роботи, виведені на екран результати перекриваються вікном інтегрованого середовища. Тимчасово прибрати це вікно і проглянути результати роботи можна, натиснувши клавіші ***Alt+F5***. Після перегляду натиснення будь-якої клавіші поверне на екран робоче поле редактора.

Меню ***Run*** містить декілька команд: їх призначення, а також пов'язані з ними клавіші приведені табл. 2.14. Тут же перераховані деякі інші команди і клавіші, що відносяться до запуску і відладки програм.

У меню ***Run*** є команда, що заслуговує окремої згадки, — це команда ***Parameters***. Якщо програмі для виконання потрібні параметри, ці параметри можна задати у вікні ***Parameters***. Після цього при кожному новому запуску програми з інтегрованого середовища параметри не доведеться вводити наново.

2.4 Практичні роботи до розділу

Практична робота №3

Програмування алгоритмів лінійної структури

Завдання 1: включіть інтегроване середовище Turbo Pascal, запустивши на виконання ярлик програми на Робочому столі. За допомогою команди ***File > New*** створіть файл з ім'ям ***proba.pas.*** У створеному файлі наберіть текст наступної програми:

```
Program First; { заголовок}
Clrscr; { очищення екрану}
Var  a,b: integer; { розділ змінних}
c: real;
BEGIN
readln (a,b);{розділ операторів}
c:=(a+5)/b;
writeln(c);
readln;
END.{ кінець програми}
```

Запустіть програму на виконання.

Завдання 2: трикутник заданий координатами своїх вершин x_1, y1, x2, y2. x3, y3. Складіть програму обчислення площі трикутника по формулі Герона при x_l = l; y_1=2; x_2=6;y_2=9; x_3= 10; y_3= 15. Складіть алгоритм і програму і виконайте її на ЕОМ. На друк виведіть значення даних, що вводяться, і результати обчислень, супроводжуючи їх найменуваннями змінних і коментарями. Модифікуйте програму так, щоб введення значень з клавіатури здійснювалося відповідно до підказки, значення яких змінних треба вводити.

Приклад виконання роботи

Введемо позначення: x_1, y_1, x_2 y_2, x_3, y_3 – координати вершин трикутника:

a, b, c – довжина сторін трикутника;

p – півпериметр трикутника:

s – площа трикутника.

Оскільки значення x_l, y_l, x_2, y_2, x_3, y_3 відомі, то їх можна ввести в розділ констант.

a, b, c, p, s – опишемо в розділі змінних як ***real.***

Формули для розрахунку змінних **a, b, з, p, s** мають наступний вигляд:

$a{:=}\sqrt{(x2-x1)^2+(y2-y1)^2};$

$b{:=}\sqrt{(x3-x2)^2+(y3-y2)^2};$

$c{:=}\sqrt{(x3-x1)^2+(y3-y1)^2};$

$p{:=}(a+b+c)/2;$

$s{:=}\sqrt{p(p-a)(pc-b)(p-c)};$

формула Герона має вигляд : S = $\sqrt{p(p-a)(p-b)(p-c)}$.

Програма рішення задачі на мові Паскаль матиме вигляд:

```
PROGRAM UPR1;
Uses crt;
const x1 = 10; y1 = 15;(*Координати першої вершини треугольника*)
x2= 15; y2=20; (*Координати другої вершини треугольника*)
x3=25; y3=30; (*Координати третьої вершини треугольника*)
var
a, b, з: real; (*Довжина сторін треугольника*)
p: real; (*Півпериметр треугольника*)
s: real; (*Площа треугольника*)
```

```
Begin
Clrscr;
a:=sqrt(sqr(x2-x1 )+sqr(y2-y1)); (*Обчислення довжини сторони три-
кутника a*)
b:=sqrt(sqr(x3-x2)+sqr(y3-y2)); (*Обчислення довжини сторони трику-
тника Ь*)
c:=sqrt(sqr(x3-x1 )+sqr(y3-y1); (*Обчислення довжини сторони трику-
тника c*)
p:=(a+b+c) /2;
(*Обчислення підлозі периметра треугольника*)
s:=sqrt(p*(p-a)*(p-b)*(p-c));
Writeln:
Write('При   координатах:   xl=',xl:2,'   ':3,'yl=',yl:2,'   ':3,
'x2=',x2:2,' ':3,'y2=',y2:2' ':3,'x3=',x3:2 ' ':3,'y3=',y3:2);
Writeln('Обчислення площі трикутника');
Write('a=',a: 4:2,' ':3,'b=',b: 4:2,' ':3,c=',c: 4:2,' ':3, 'p=',p:
4:2,'':3,'s=',s: 6:3);
Readin;
End.
```

Перший оператор виведення ***WRITELN*** друкує текст на екрані з першої позиції рядка. Виводить значення координат по формату 2:0 через три пропуски. Другий оператор виводить на екран текст і готує виведення інформації з нового рядка. Третій оператор друкує позначення сторін трикутника, півпериметра по формату 4:2, а площі – 6:3, виводить їх значення на екран, відокремлюючи їх один від одного трьома пропусками.

В результаті на екрані одержимо повідомлення:

При координатах: xl=10 yl=15 x2=15 y2=20 x3=25 y3=30

Обчислення площі трикутника

a=7.07 b=14.14 c=15.80 p=18.50 s= 50.137

Скоректуємо програму:

```
PROGRAM UPR2;
Uses crt;
var
xl,yl,x2,y2,x3,y3: real; (*Координати вершин треугольника*)
a, b, з : real; (*Довжина сторін треугольника*)
p: real; (*Півпериметр треугольника*)
s: real; (*Площа треугольника*)
Begin
clrscr;
Writeln('Введіть координати вершин трикутника');
Writeln('x1 = 10, y1 = 15, x2= 15, y2=20, x3=25, y3=30');
Readln(x1,y1,x2, y 2,x3,y3); a:=sqrt(sqr(x2-x1)+sqr(y2-y1));
(*Обчислення довжини сторони трикутника a*);
```

```
b:=sqrt(sqr(x3-x2)+sqr(y3-y2)); (* Обчислення довжини сторони три-
кутника Ь*);
c:=sqrt(sqr(x3-x1 )+sqr(y3-y1); (*Обчислення довжини сторони трику-
тника з *);
p:=(a+b+c) /2; (*Обчислення півпериметра треугольника*);
s:=sqrt(p*(p-a)*(p-b)*(p-c));
Writeln;
Write('При     координатах:     xl=',xl:2,":3,'yl=',yl:2,":3,'x2=',
x2:2,":3, 'y2=',y2:2,":3, 'x3=',x3:2,":3,'y3=',y3:2);
Writeln('Обчислення                площі                трикутника');
Write('a=',a:4:2,":3,'b=',b:4:2,":3,'c=',c:4:2,":3,
'p=',p:4:2,":3,'s=',s:6:3);
Readln;
End.
```

Завдання для самостійної роботи

1. Складіть програму лінійного алгоритму дій медичної сестри при астматичному статусі згідно наступної текстової інструкції:

1) дати розпорядження молодшій медичній сестрі негайно викликати чергового лікаря, лікаря-анестезіолога.

2) терміново транспортувати хворого до відділення інтенсивної терапії.

3) за розпорядженням лікаря виконати: киснева терапія (35-45% кисню у повітряній суміші); інфузійна терапія: 3-3,5 (Гемодез, Поліглюкін); приготувати препарати і під керівництвом лікаря вводити: Еуфілін; Преднізолон 60-90 мл.

2. Складіть програму лінійного алгоритму дій медичної сестри при нападі асфіксії згідно наступної текстової інструкції:

1) дати розпорядження молодшій медичній сестрі — негайно викликати чергового лікаря, лікаря-анестезіолога.

2) забезпечити доступ пацієнта до свіжого повітря.

3) попросити пацієнта затримати подих.

4) інгаляція Сульбутомолу (1-2 вдихи), особам літнього віку — Атровент.

5) спостерігати за диханням, пульсом, артеріальним тиском пацієнта, а також кашлем, появою мокротиння.

6)за розпорядженням лікаря: приготувати препарати і у присутності лікаря вводити внутрішньовенно: Еуфілін 2,4% — 10 мл; подати для інгаляції зволожений кисень.

3. Складіть програму лінійного алгоритму обчислення потенційної енергії тіла.
4. Складіть програму лінійного алгоритму обчислення кінетичної енергії тіла.
5. Складіть програму лінійного алгоритму обчислення площі і периметра квадрату.
6. Складіть програму лінійного алгоритму дій медичної сестри при гіпертонічному кризі згідно наступної текстової інструкції:

1)зручно укласти.

2) гіпотензивні препарати: магнезія ($MgSO_4$ 25%) 10 мл. в / в або в / м.

3) дибазол 1% до 5 мл. + Папаверин до 2 мл. + Анальгін до 2 мл.

4) для профілактики інфаркту міокарда під язик нітрогліцерин.

5) відволікаюча терапія: баночний або гірчичний комір, гірчичники на литкові м'язи, теплі ванни для ніг.

6) гірудотерапія: п'явки на соскоподібного відросток (за вухом). П'явки виділяють гірудин, який перешкоджає згортанню крові і освіти тромбозу.

7) джгути на кінцівки.

8) лазикс 40-80-мг.

9) в / в на фіз. розчині 0,01% розчин клофеліну.

10) в / м еуфілін.

7. Складіть програму лінійного алгоритму проведення медичних маніпуляцій при дослідженні звуків дихання, що проводяться за допомогою комплекса «КоРа – 03 М1», в якому трансформація звукових образів у візуальні дозволяє ефективно та достовірно об’єктивізувати аускультаційні показники, що характеризують певний вид бронхолегеневого захворювання.
8. Складіть програму лінійного алгоритму проведення контролю якості дезінфекції виробів медичного призначення згідно наступної текстової інструкції:

1) промивання проточною водою після дезінфекції над раковиною протягом 30 секунд до повного знищення запаху деззасобів;

2) замочування в миючому розчині при температурі води 50 ° С на 15 хвилин шпрі-ців і головок в розібраному стані;

3) миття кожного виробу в цьому ж розчині, де проводилось замочування, з по-міццю йоржа або ватного тампона протягом 30 секунд;

4) споліскування проточною водою (після миючого засобу «Біолот» - 3 хвилини; після розчинів перекису водню з миючим засобом - 5 хвилин);

5) споліскування дистильованою водою протягом 30 секунд;

6) просушування гарячим повітрям при температурі +75 .. + 87 в сушильних шафах.

Практична робота № 4

Організація розгалуження у програмі

Завдання 1. Користуючись умовним оператором IF обчислити значення:

$$y = \begin{cases} \sin(x), \textit{ якщо } 1 \le x \prec 2; \\ e^{-x}, \quad \textit{якщо } 2 \le x \prec 3; \\ \ln(x), \quad \textit{якщо } 3 \le x \prec 4; \\ tg(x), \quad \textit{якщо } 4 \le x \prec 5. \end{cases}$$

Приклад виконання завдання

```
Program UPR3;
Var
x, y: real;
n: integer;
Begin
     writeln('Введіть число x');
     read (x);
     if (x<1) or (x>=5) then
          writeln('x не належить ', 'даної області')
     else
          begin
          n := trunc(x);
          case n of
               1: y := sin(x);
               2: y := exp(-x);
               3: y := ln(x);
               4: y := sin (x) /cos(x);
          end;
     writeln('y=' y:6:2);
     end;
End.
```

Завдання 2. Користуючись умовним оператором IF з трьох дійсних чисел ***x***, ***y***, ***z***. обрати максимальне з їх.

Приклад виконання завдання

```
Program UPR4;
var x,y,z,max: real;
BEGIN
     write('Введіть числа x, y, z через пробіл');
     readln(x,y,z);
     if x>y then max:=x else max:=y;
     if z>max then max:=z;
     writeln ('Максимальне з їх ',max);
END.
```

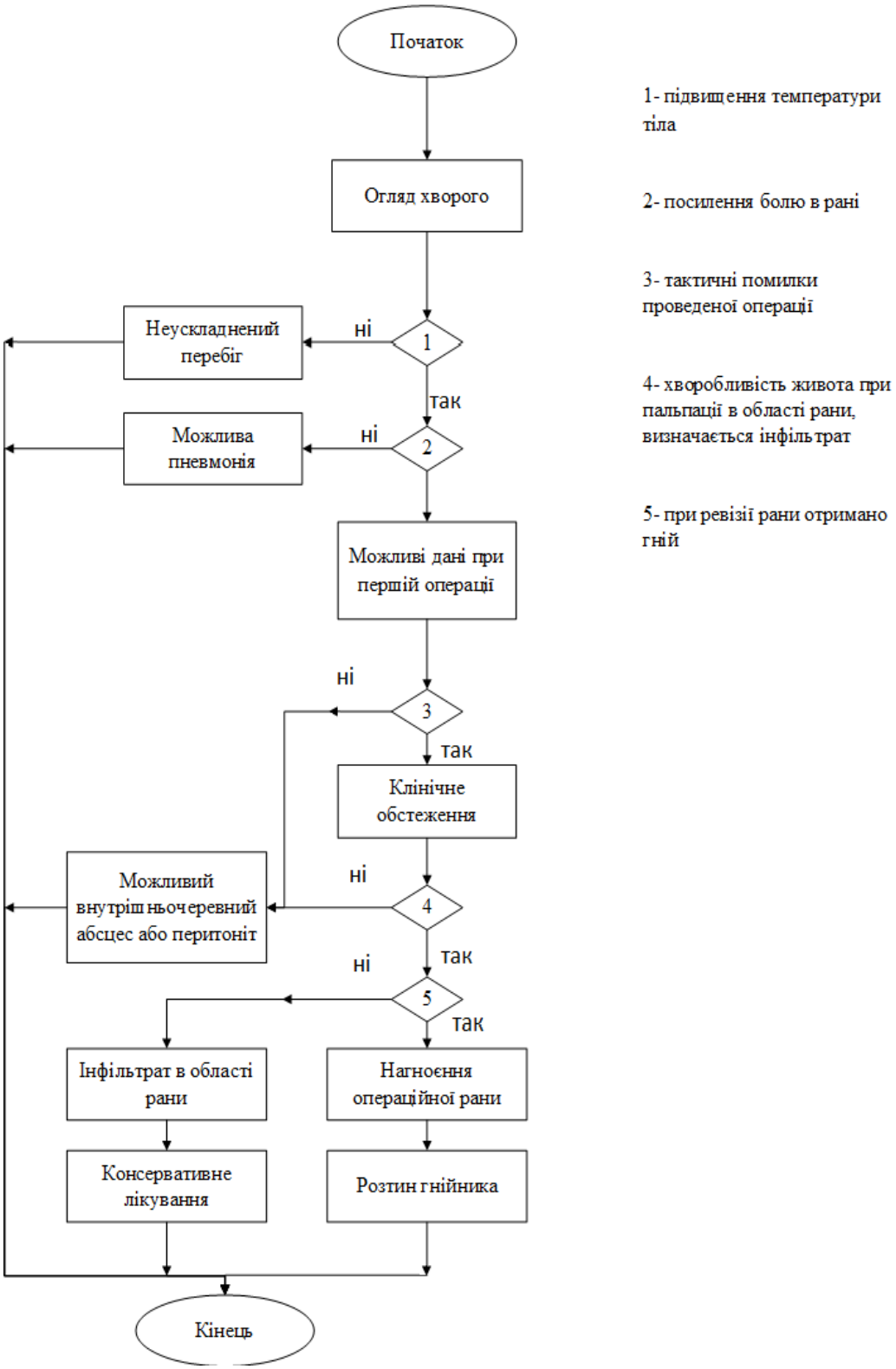
Завдання 3. Відомі дійсні позитивні числа ***x***, ***y***, ***z***. Користуючись умовним оператором IF з'ясувати, чи існує трикутник з довжинами сторін ***x***, ***y***, ***z i*** якщо існує, то визначити його вигляд по кутах.

Приклад виконання завдання

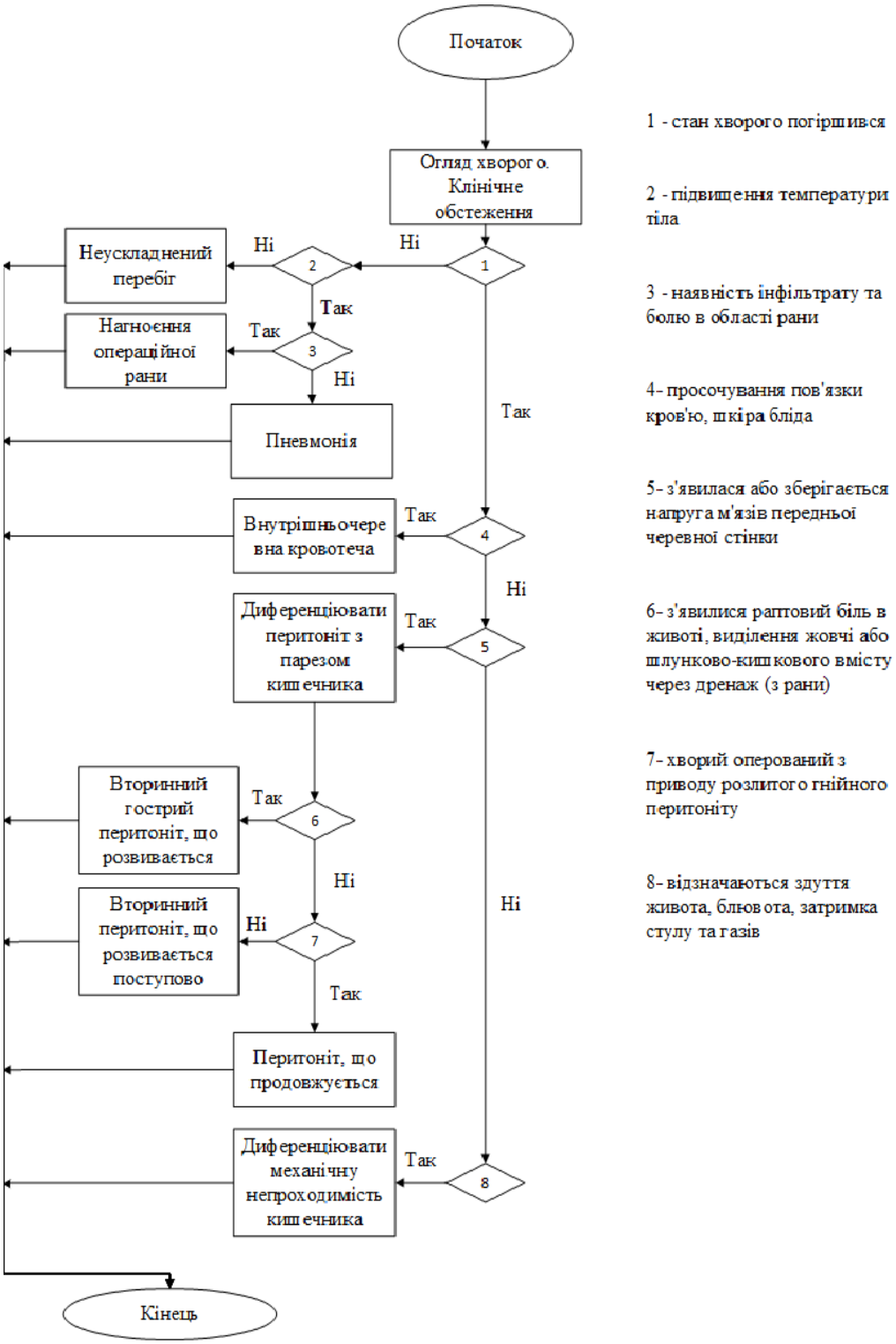
```
Program UPR5;
var x,y,z,max,d: real;
BEGIN
     write('Введіть довжини сторін трикутника через пропуск');
     read(x,y,z);
     if x>y then max:=x else max:=y;
     if z>max then max:=z;
     write('Трікутник');
     if 2*max<x+y+z then
          begin
          d:=sqr(x)+sqr(y)+sqr(z)-2*sqr(max);
          if d>0 then write('гострокутний');
          if d=0 then write('прямокутний');
          if d<0 then write('тупокутний')
          end
     else
     write('не існує!');END.
```

Завдання для самостійної роботи

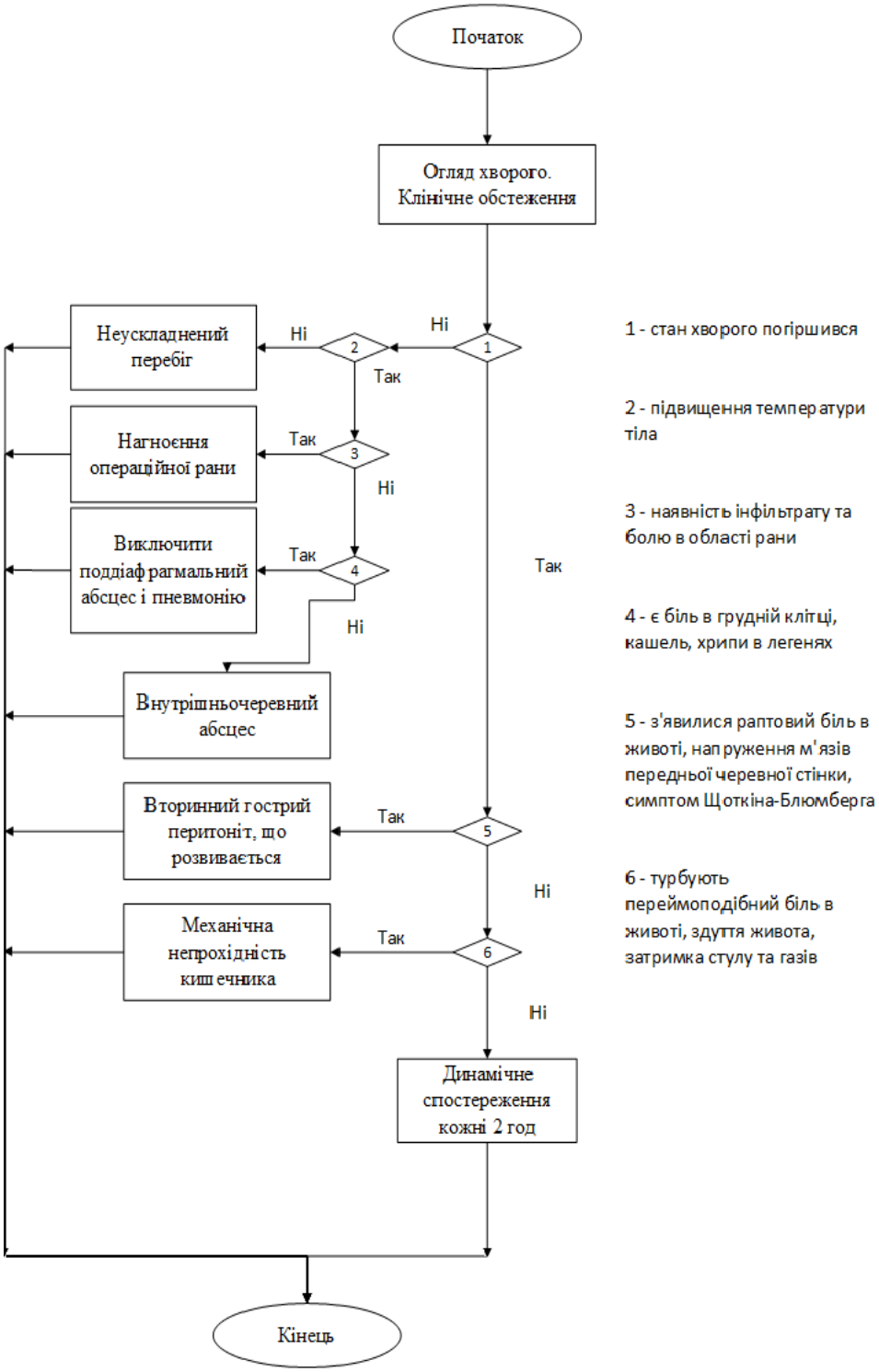
1. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою медичної задачі:



2. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою діагностичного пошуку прийняття рішень при виявленні ускладнень в перші 5 діб після операції:

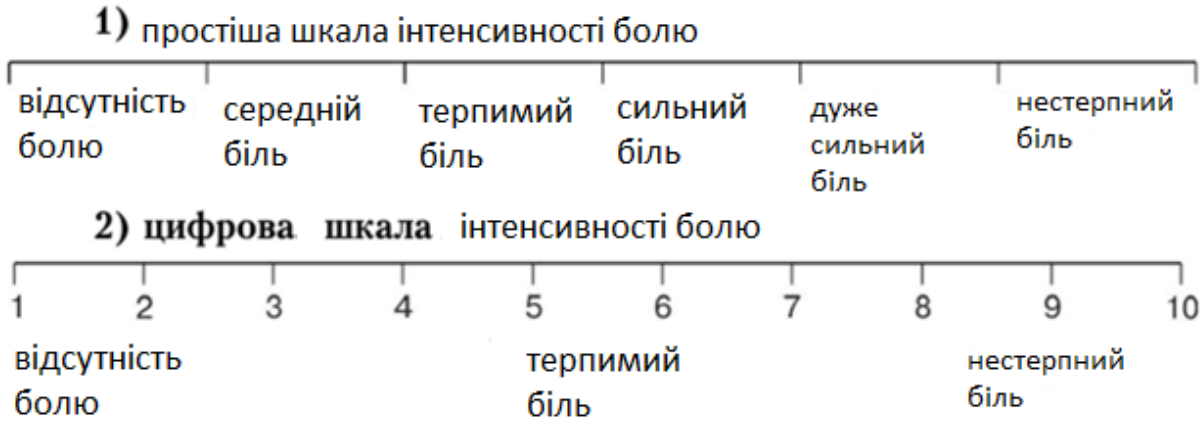


3. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою діагностичного пошуку прийняття рішень при виявленні ускладнень на 6-12 добу після операції:

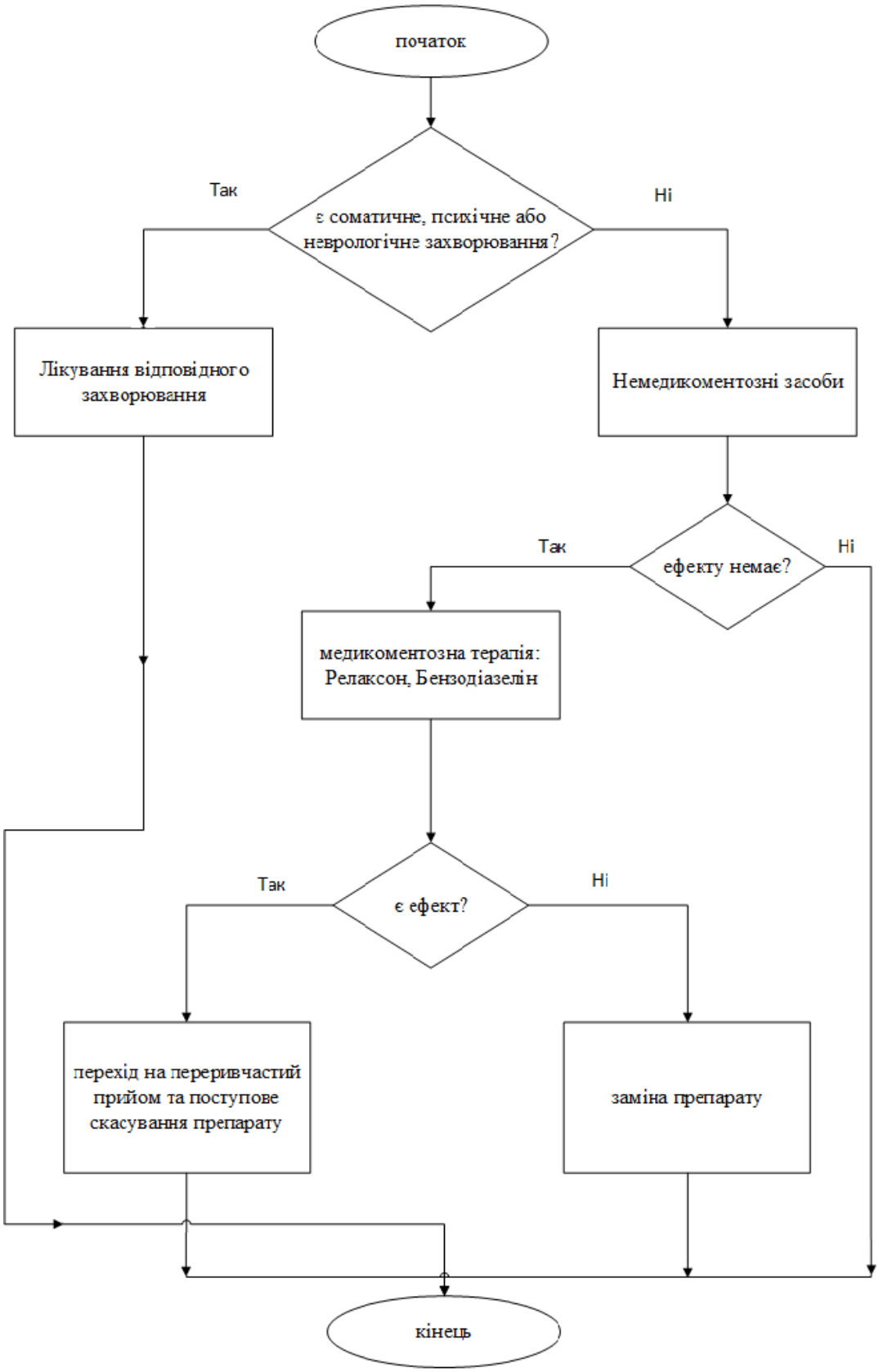


4. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою медичної задачі:

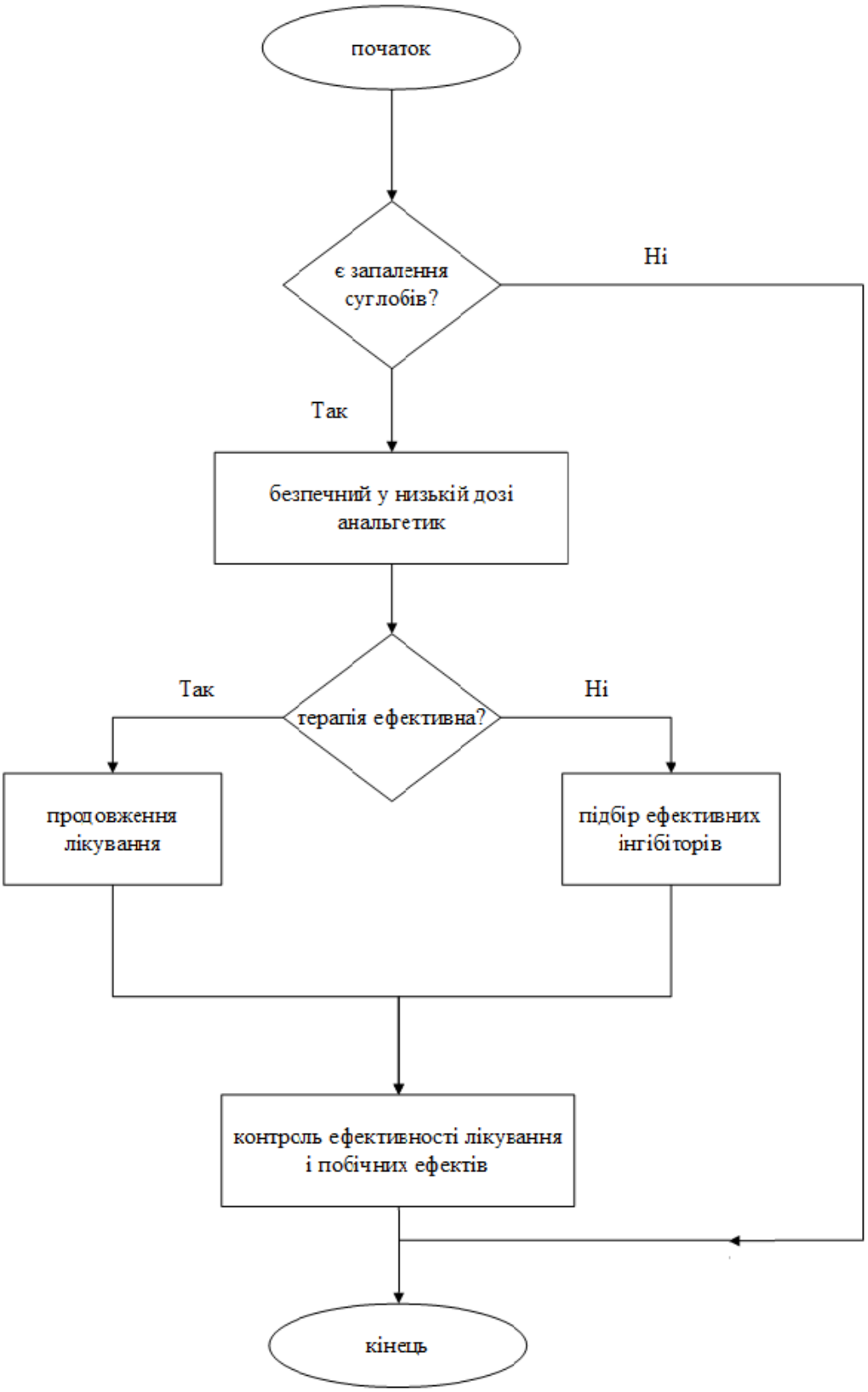
5. Користуючись умовним оператором IF скласти програму визначення ступеню болю згідно наступної цифрової шкали



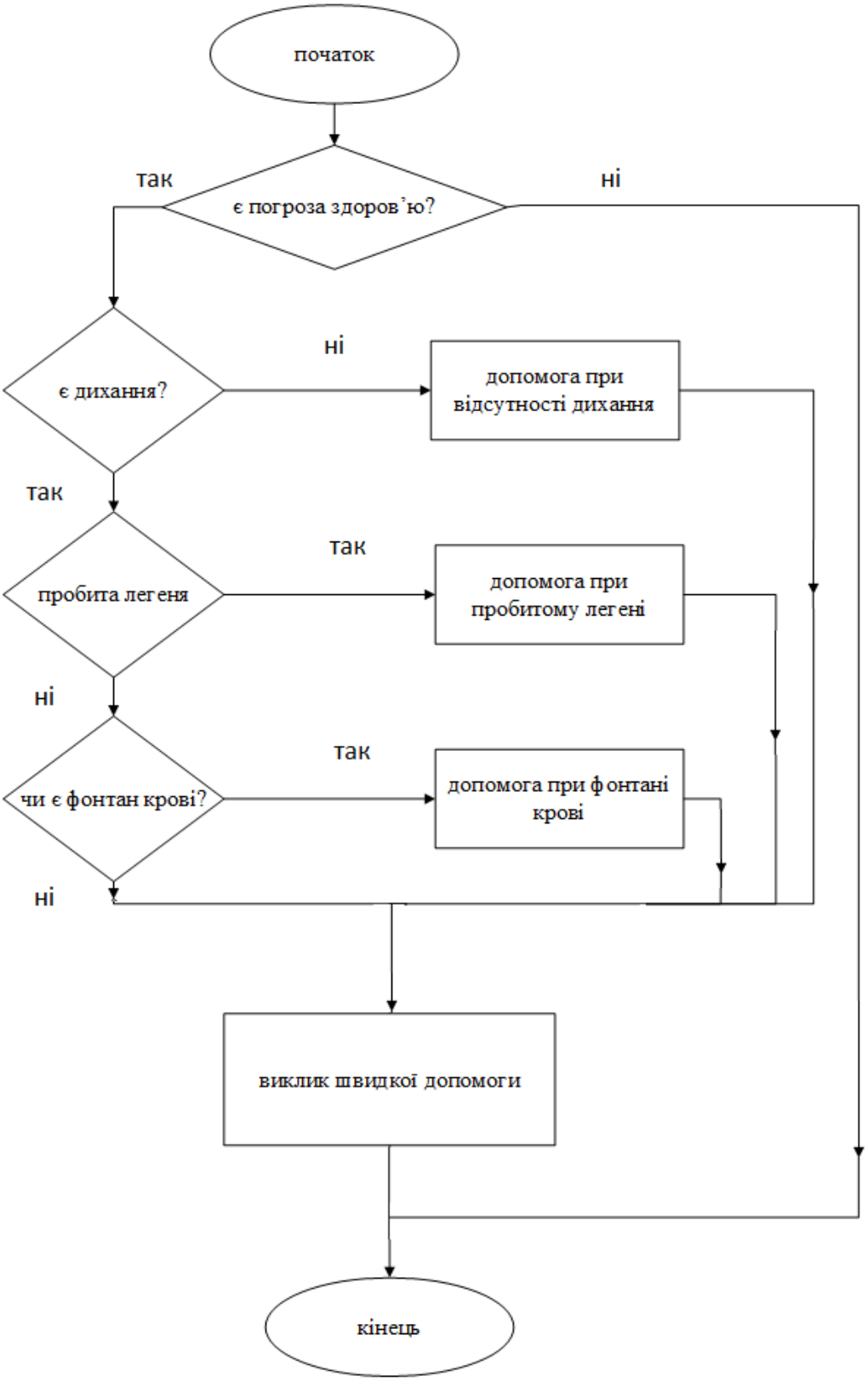
6. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою медичної задачі:



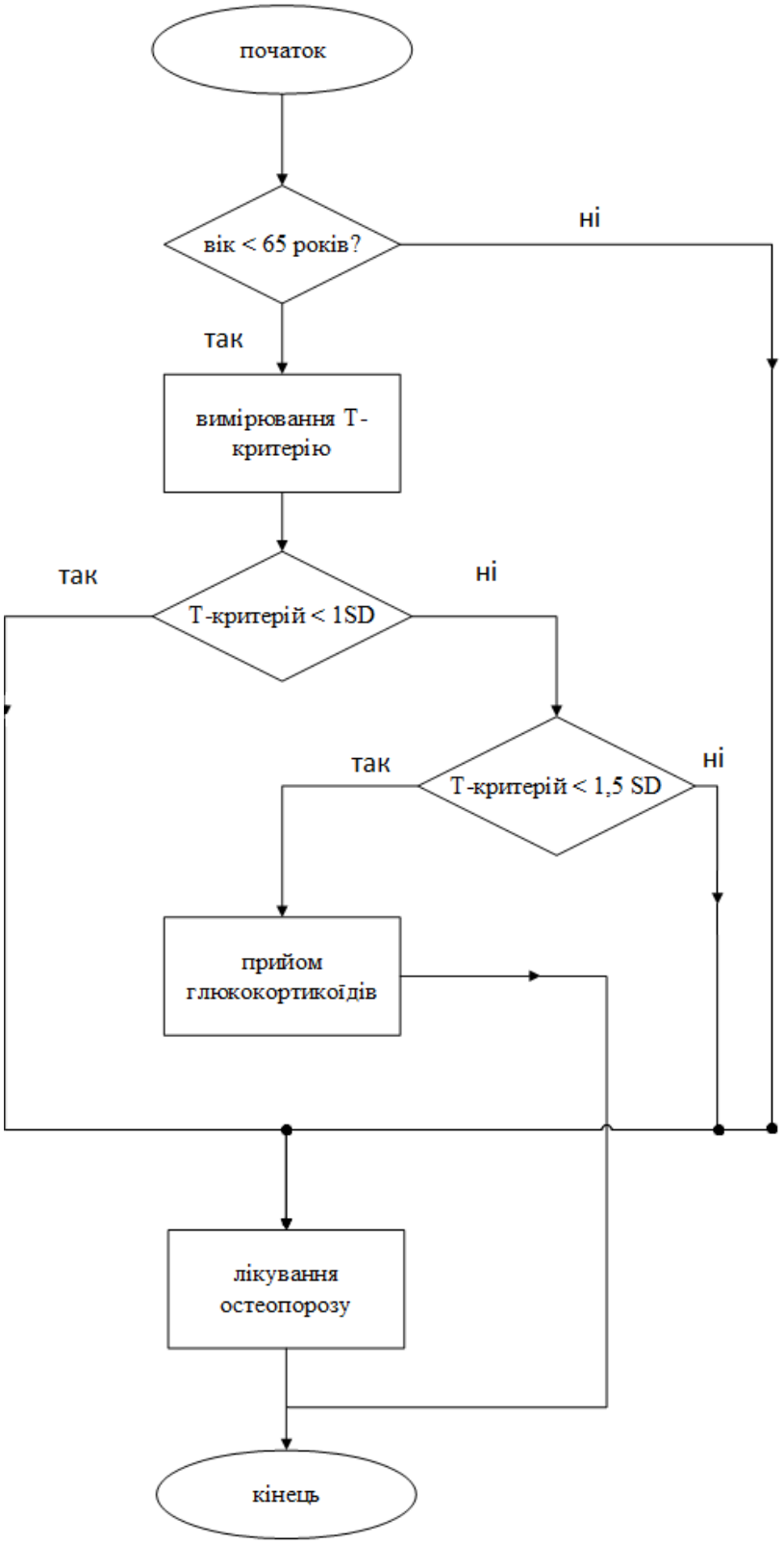
7. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою фармацевтичної задачі:



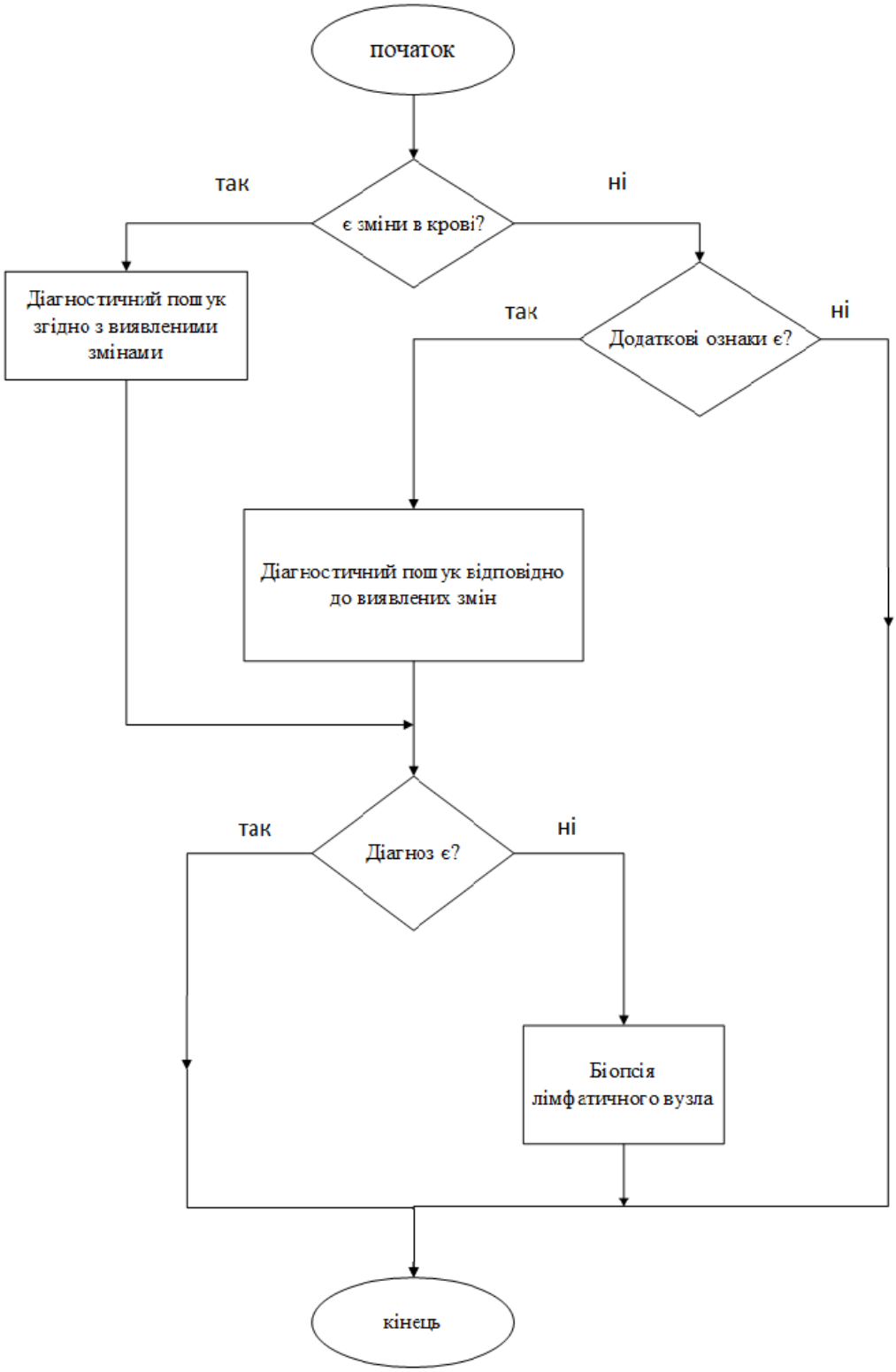
8. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою фармацевтичної задачі:



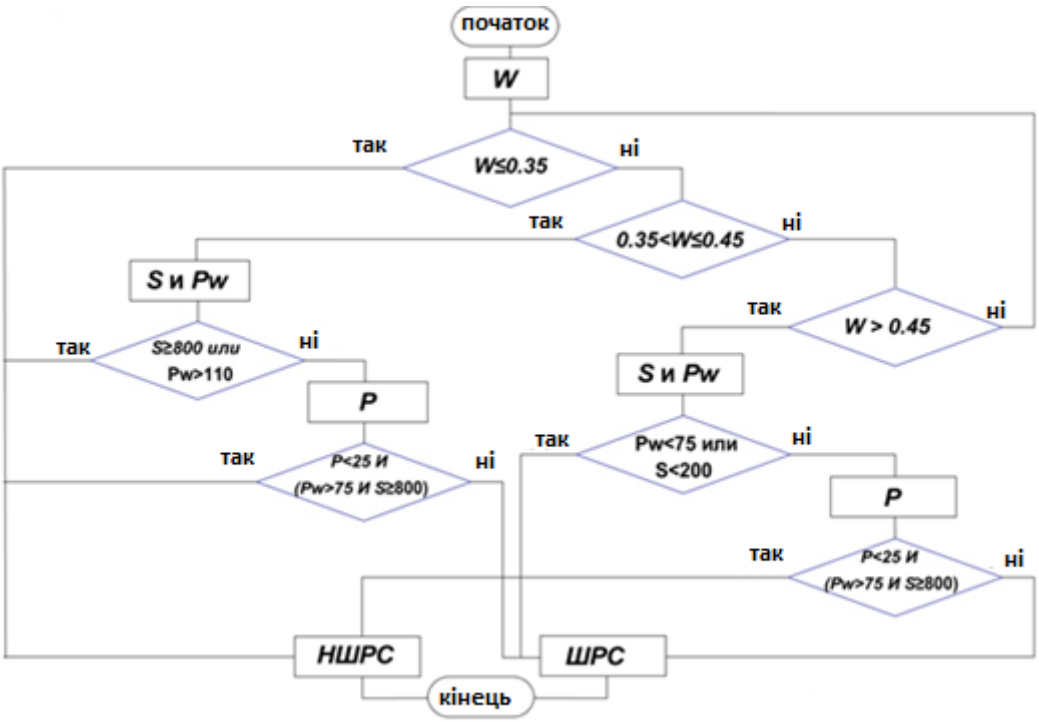
9. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою фармацевтичної задачі:



10. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою фармацевтичної задачі:



11. Користуючись умовним оператором IF скласти програму згідно з наданою блок-схемою фармацевтичної задачі:



Практична робота № 5

Організація циклів у програмі

Завдання 1. Дано дійсне число. Візначити, чи утворюють цифри числа симетричну послідовність, тобто читаються однаково зліва направо і справа наліво Прикладом симетричної послідовності є число 012210.

```
Program UPR6-1
var m,n,x: integer;
BEGIN
     repeat
          write('Введіть натуральне число число ->');
          readln(n);
     until n>0;
     m:=n; x:=0;
     while n>0 do
          begin
          x:=x*10+n mod 10;
          n:=n div 10;
          end;
     if x=m then writeln('Паліндрон')
     else writeln('Не палиндром');
END.
```

Завдання 2. Вівести на екран числа Фібоначчі від 1 до ***n***. Кожне число послідовності з чисел Фібоначчі, починаючи з третього, виходить складанням двох попередніх чисел. Напріклад: 1,1,2,3,5,8,13,21...

```
Program UPR6-2
var a,b,c,i,n: integer;
BEGIN
     write('Введіть число  N -> ');
     readln(n);
     a:=1; b:=1;
     write(a,' ',b);
     i:=3;
     repeat
          c:=a+b;
          write(' ',c);a:=b;
          b:=c; i:=i+1;
     until i>n;
END.
```

Завдання 3. Дано натуральне число n. Візначиті, чи є n числом Фібоначчі, тобто чи є воно елементом послідовності 1, 1, 2, 3, 5, 8, 13, 21, 34 …

```
Program UPR6-3
var n,a,b,c: integer;
BEGIN
     repeat
          write('Введіть натуральне число  ->');
          readin(n);
     until n>0;
     a:=1; b:=1; c:=1;
     while c<n do
          begin
          c:=a+b;
          a:=b;
          b:=c;
          end:
     if c=n then writeln('Число Фібоначчі') else writeln('Не
число Фібоначчі');
END.
```

Завдання 4. На інтервалі [2; n] знайти натуральне число з максимальною сумою дільників.

```
Program UPR6-4
var n,i,sum_max,sum,k,ch: integer;
BEGIN
     write('Введіть число n ->');
     readln(n);
```

```
        sim_max:=1; ch:=1;
        for i:=2 to n do
             begin
                  sum:=0;
                  for k:=1 to i div 2 + 1 do
                  if i mod до = 0 then sim:=sum+k;
                  sum:=sum+i;
                  if sum>sum_max then
                  begin
                  sum_max:=sum;
                  ch:=i;
                  end;
             end;
        writeln('Максимальну торбу дільників ',sum_max,' має чи-
        сло',ch)
END.
```

Завдання для самостійної роботи

1. Користуючись операторами циклу складіть програму вирішення наступної задачі: динаміка концентрації амфітаміну у плазмі крові за певного рівня кислотності РН описується рівнянням

$$C_{амфетамина}(t) = e^{\frac{-0,693 \cdot t}{7 \cdot ph - 37,5}},$$

де С – концентрація препарату; е- час після введення препарату; РН = 6.5 - кислотність. За умови, що пацієнтові було введено амфетамін у час t = 0, визначте, як змінювалась концентрація в плазмі крові амфетаміну кожні 10 хвилин протягом 2 годин.

2. Користуючись операторами циклу складіть програму вирішення наступної задачі: динаміка концентрації метілфенідату гідрохлориду у плазмі крові за певного рівня кислотності РН описується рівнянням

$$C_{метилфенидата}(t) = e^{-0,365 \cdot t}$$

де С – концентрація препарату; е- час після введення препарату; РН = 6.5 - кислотність. За умови, що пацієнтові було введено препарат у час t = 0, визначте, як змінювалась концентрація в плазмі крові метілфенідату кожні 15 хвилин протягом 3 годин.

3. Користуючись операторами циклу складіть програму вирішення наступної задачі: даний каскад з N послідовних нейронів, що передають сінаптичний сигнал. Сигнал передається від одного нейрона іншому з однаковою затримкою у часі τ = 0,5с. В цьому випадку швидкість розповсюдження сигналу R як функція від часу t (у секундах) задається формулою:

$$R(t) = \frac{1}{\tau \cdot 10!} \cdot \left(\frac{t}{\tau}\right)^N \cdot e^{\frac{-t}{\tau}}$$

Визначте , як змінюється швидкість розровсюдження сигналу кожну 1 с.

4. Користуючись операторами циклу складіть програму вирішення наступної задачі: зростання кількості бактерій у популяції описується рівнянням:

$$f(t) = \frac{a}{1 + b \cdot e^{-a \cdot t}},$$

де f – кількість бактерій у популяції (в тисячах), t – час розвитку популяції (дні), a – гранична кількість бактерій у популяції (в тисячах), b – параметр, пов'язаний зі швидкістю розмноження бактерій. Визначте, як змінюється популяція кожен день на протязі 10 днів, якщо a=1,2, b=15,33.

5. Користуючись операторами циклу складіть програму вирішення наступної задачі: зростання популяції мурашок описується рівнянням

$$P(t) = \frac{P_L}{1 + c \cdot e^{-k \cdot t}},$$

де P – розмір популяції; t – час розвитку популяції (місяці); P_L – гранична кількість комах у популяції; c та k – параметри, що характеризують умови виживаності мурашника. Визначте, як змінюватиметься популяція протягом року через кожен місяць при k=1,5, c=0,99 і P_L=4000.

6. Користуючись операторами циклу складіть програму вирішення наступної задачі: формула, що описує проникність клітинної мембрани по відношенню до іонів натрію, виглядає так:

$$pNa(t) = pNa_{зовн} \cdot m^3(t) \cdot h(t) + INa\,.$$

де t – час (у мілісекундах), $pNa_{зовн}$ – проникність мембрани зовні клітини для іонів натрію, INa – канали у мембрані для іонів натрію, $m(t) = 1 - e^{\frac{-t}{\tau_m}}$, $h(t) = e^{\frac{-t}{\tau_h}}$ – рівняння кінетики відкриття й зачинення воріт каналу мембрани по відношенню до іонів натрію, τ_m, τ_h – константи воріт каналу мембрани. Визначте, як змінюється приникність кожну 1 мс до 50 мс при наступних значеннях параметрів: τ_m=2, τ_h=0,75, INa=0,05, $pNa_{зовн}$=200.

7. Користуючись операторами циклу складіть програму вирішення наступної задачі: формула, що описує проникність клітинної мембрани по відношенню до іонів калію, виглядає так:

$$pK(t) = pK_{зовн} \cdot n^4(t) \cdot i(t) + IK,$$

де t – час (у мілісекундах), $pK_{зовн}$ – проникність мембрани зовні клітини для іонів калію, IK – канали у мембрані для іонів калію, $n(t) = 1 - e^{\frac{-t}{\tau_n}}$, $i(t) = e^{\frac{-t}{\tau_i}}$ – рівняння кінетики відкриття й зачинення воріт каналу мембрани по відношенню до іонів калію, τ_n, τ_i – константи воріт каналу мембрани. Визначте, як змінюється з часом (через 2мс) проникність по відношенню до іонів калію за наступних значень параметрів: τ_n=3, τ_i=4, IK=0,75, $pK_{зовн}$ =5.

8. Користуючись операторами циклу складіть програму вирішення наступної задачі: рівняння Гольдмана—Ходкінга—Каца для потенціалу клітинної мембрани по відношенню до її проникності іонами натрію та калію, виглядає так:

$$E(t) = -R \cdot \frac{T}{F} \cdot \ln\left(\frac{pNa(t) \cdot Na_{внутр} + pK(t) \cdot K_{внутр}}{pNa(t) \cdot Na_{зовн} + pK(t) \cdot K_{зовн}}\right),$$

де R – універсальна газова константа, R=8,314Дж/(К·моль), t – час (у мс), T – температура клітини (у Кельвінах), F – константа Фарадея, F=96500 С/моль, $Na_{зовн}$, $Na_{внутр}$, $K_{зовн}$, $K_{внутр}$ – концентрації іонів натрію та калію зовні та всередині клітини, $pNa(t)$, $pK(t)$ – проникності мембрани по відношенню до іонів натрію та калію:

$$pNa(t) = pNa_{зовн} \cdot m^3(t) \cdot h(t) + INa,$$

$$pK(t) = pK_{зовн} \cdot n^4(t) \cdot i(t) + IK,$$

де $pNa_{зовн}$, $pK_{зовн}$ – проникність мембрани зовні клітини для іонів натрію та калію, INa, IK – канали у мембрані для іонів натрію та калію, $m(t) = 1 - e^{\frac{-t}{\tau_m}}$, $h(t) = e^{\frac{-t}{\tau_h}}$ – рівняння кінетики відкриття й зачинення воріт каналу мембрани по відношенню до іонів натрію, $n(t) = 1 - e^{\frac{-t}{\tau_n}}$, $i(t) = e^{\frac{-t}{\tau_i}}$ – рівняння кінетики відкриття й зачинення воріт каналу мембрани по відношенню до іонів калію, τ_m, τ_h, τ_n, τ_i – константи воріт каналу мембрани.Визначте як змінюється з часом (кожні 2 мс) потенціал проникності мембрани по відношенню до іонів натрію та калію за наступних значень параметрів: T=310, $Na_{зовн}$=145, $Na_{внутр}$=14, $K_{зовн}$=4,5, $K_{внутр}$=157, τ_m=2, τ_h=0,75, τ_n=3, τ_i=4, INa=0,05, IK=0,75, $pNa_{зовн}$=200, $pK_{зовн}$=5.

9. Користуючись операторами циклу складіть програму вирішення наступної задачі: розвиток деякої популяції мікроорганізмів можна описати рівнянням:

$$f(t) = \frac{k}{e^{\theta(t)} + 1},$$

де f – кількість мікроорганізмів у популяції (в тисячах), t – час розвитку популяції (дні), k – гранична кількість мікроорганізмів у популяції (у тисячах),

$$\theta(t) = \frac{3}{7}t^2 - 2{,}5t + 0{,}1.$$

Визначте, як змінюватиметься популяція на протязі 20 днів (по дням) при k=5,3.

10. Користуючись операторами циклу складіть програму вирішення наступної задачі: динаміка концентрації амфетаміну в плазмі крові при певному рівні кислотності ph описується рівнянням:

$$C(t) = e^{\frac{-0{,}693 \cdot t}{7 \cdot ph - 37{,}5}},$$

де C – концентрація препарату, t – час після введення препарату (години), ph – кислотність. Визначте, як спадає концентрація амфетаміну в плазмі крові кожні 0,5 години при ph=6,5.

11. Користуючись операторами циклу складіть програму вирішення наступної задачі: сила струму іонів кальцію на поверхні клітинної мембрани описується формулою:

$$I_{Ca}(v) = P_{Ca} \cdot 2 \cdot v \cdot F \cdot \frac{Ca_{внутр} - Ca_{зовн} \cdot e^{-v}}{1 - e^{-v}},$$

Де I_{Ca} – сила струму іонів кальцію, v – напруга, P_{Ca} – проникність мембрани по відношенню до іонів кальцію, $Ca_{внутр}$, $Ca_{зовн}$ – щільність іонів кальцію всередині та зовні клітини (кількість іонів на одиницю площі поверхні клітинної мембрани),

$$F = F(v) = \frac{v \cdot R \cdot T}{2 \cdot V_m},$$

де V_m – одинична напруга частки активації, R – універсальна газова константа, R=8,314 Дж/(К·моль), T – температура клітини (у Кельвінах). Визначте, як змінюється значення струму із зміною напруги від 0,01В до 0,25В (з кроком 0,01В) , якщо P_{Ca}=0,3, $Ca_{внутр}$=0,002, $Ca_{зовн}$=0,00005, V_m= –20, T=310.

2.6. Контрольні питання до розділу 2

1. Що таке алфавіт мови Паскаль?
2. Що таке ідентифікатор ?
3. Що таке оператор ?
4. На які групи ділять всіх операторів в Паскалі?
5. Що таке змінна?
6. Що таке тип даних?
7. За допомогою якого оператора змінна набуває значення?
8. Як записати на мові Паскаль зведення числа X в ступінь A?
9. . Як записати на мові Паскаль *lg(123)?*
10. Перерахуйте прості типи даних в Паскалі
11. Завдання яких чисел забезпечує цілий тип даних?
12. Запишіть на мові Паскаль дійсне число *0,00145* як число з плавучою крапкою.
13. У якому вигляді можливий запис дійсного числа на мові Паскаль?
14. Що є дані символьного типа?
15. Які 2 значення можуть приймати змінні типу *Boolean?*
16. Чи можна змінювати тТип змінної в тексті програми на мові Паскаль і чому?
17. Що таке мантиса і порядок у записі дійсного числа?
18. За допомогою яких операцій здійснюється введення даних на мові Паскаль?
19. За допомогою яких операцій здійснюється виведення даних на мові Паскаль?
20. Що таке форматний вивід даних?
21. Дійсне число A=5,321 виводиться на екран командою WRITE(A:4:3). Чому рівна точність представлення числа A?
22. Дійсне число B=12,345 виводиться на екран командою WRITE(B:5:3). Чому рівна ширина поля виведення числа B?

23. Дійсне число C=567,89 виводиться на екран командою WRITELN(C). Як воно виглядатиме на екрані?

24. Для чого призначений оператор привласнення?

25. Запишіть синтаксичну конструкцію для оператора циклу з передумовою.

26. Запишіть синтаксичну конструкцію для оператора циклу з умовою поста.

27. Запишіть синтаксичну конструкцію для оператора умовного

28. Призначення умовного оператора.

29. Запишіть синтаксичну конструкцію для оператора вибору.

30. Який оператор використовується, якщо число повторень циклу відоме перед його початком?

31. Коли використовується службове слово DOWNTO в циклі з параметром?

32. Якого типа повинні бути дані, задаючі параметр циклу і діапазон його зміни?

33. Для чого використовується оператор безумовного переходу?

34. Чому рівний результат дії математичної функції *9 mod 2?*

35. Результат дії математичної функції *9 div 2* рівний

36. Чому рівний результат дії математичної функції *Int(17,53)*?

37. Чому рівний результат дії математичної функції *Frac(23,173)?*

38. Чому рівний результат дії математичної функції *Sqrt(144)*?

39. Чому рівний результат дії математичної функції *Sqr (16)?*

40. Використовуючи текст програми

```
CASE (x+2) OF
     2: y:=x+1;
     4: y:= x-2;
     ELSE WRITELN( 'ERROR' );
END
```

визначити, чому дорівнює У, якщо Х= 0; 1; 2; 3.

3. СТРУКТУРОВАНІ ТИПИ ДАНИХ. ПОДПРОГРАМИ І МОДУЛІ

У програмуванні на мові Turbo – Pascal використовують структуровані типи даних. До них відносяться:

- масив;
- записи;
- множина;
- файл.

Структурований тип оголошується в розділі описів після зарезервованого слова TYPE.

3.1. Масиви

Масив є впорядкованою безліччю однотипних елементів. У Turbo – Pascal масив описується змінною складної структури. При описі необхідно вказати:

- спосіб об'єднання елементів в структуру;
- число елементів;
- тип елементів**.**

Загальний вид опису масиву:

<им'я масиву>: ARRAY [тип – діапазон індексів] OF <тип елементів масива>;

Доступ до кожного елементу масиву здійснюється за допомогою індексів.

Тип – діапазон кожного індексу задається лівою і правою межами зміни індексу. Число індексів визначають структуру масиву: якщо використовується один індекс, то такий масив називається ***одномірним***, якщо два індекси – ***двомірним***. У загальному випадку розмірність масиву може бути довільною.

3.1.1. Одномірні масиви

У математиці одномірному масиву відповідає вектор, наприклад:

$$\overline{x} = \{x_i\}; i = \overline{1, n}$$

де

x_i – компоненту (координата) вектора;

i - номер компоненти;

n - число компонент.

У Турбо Паскалі опис одномірного масиву задається таким чином:

```
<им'я масива>: ARRAY [m1 ...m2] OF < тип элементів>;
```

Індекс одномірного масиву визначає порядковий номер елементу в масиві, а m1 …m2 – діапазон значень індексу.

Тип елементів в масиві може бути будь-яким: простим (REAL, INTEGER, CHAR), структурним (ARRAY), строковим (STRING).

По опису масиву в пам'яті комп'ютера виділяється область послідовних осередків, в яку при виконанні програми заносяться значення елементів масиву. Наприклад, по опису

```
Var       X: array [15] of real;
```

виділятиметься область з п'яти послідовних осередків для запису значень елементів речовинного типа.

Введення – виведення масивів здійснюються поелементно за допомогою оператора циклу ***FOR***, в якому як параметр використовується індекс.

Приклад 1. Організувати введення з клавіатури масиву:

$$A = (1.2; 5; -6.8; 14).$$

У розділі опису змінних необхідно задати опис масиву і індексу.

```
Var
A: array [14] of real;
i: integer;
```

У операторній частині програми рекомендується введення масиву організувати у вигляді діалогу, що супроводжує введення відповідним повідомленням на екрані.

```
Begin { початок програми}
writeln('Введіть масив A');
for i:= 1 to 4 read(a[i]);
```

На клавіатурі набирається: **1.2 < *Enter* > 5 < *Enter* > 6.8 < *Enter* > 14 < *Enter* >.**

Приклад 2. Організувати виведення масиву А на екран так, щоб всі елементи розташовувалися на одному рядку екрану.

У програмі треба записати наступні оператори:

```
for i:= 1 to 4 do write(a[i]:5:1);
writeln;
```

Вид масиву, що виводиться, на екрані: **1.2 < *Enter* > 5.0 < *Enter* > -6.8 < *Enter* > 14.0 < *Enter* > .**

Оператор WRITELN без списку служить для переходу курсора до початку наступного рядка.

3.1.2. Обробка одномірних масивів

При рішенні задач обробки масивів використовують базові алгоритми реалізації циклічних обчислювальних процесів: організація лічильника, накопичення сум і добутків, пошук мінімального і максимального елементів масиву.

Приклад 1. Організація лічильника.

Визначити кількість елементів заданого цілочисельного масиву $B = \{b_i\}; i = \overline{1,20}$, які діляться на 3 без залишку.

```
Program OM_1;
Var
B: array [1. 20] of integer;
i, L: integer;
Begin
     writeln('Введіть масив B');
     for i :=1 to 20 do read(b[i]);
     L := 0;
     for i :=1 to 20 do
     if  (b[i] MOD 3) = 0 then L := L+1;
     writeln('L =', L);
End.
```

Приклад 2. Накопичення суми і добутку.

Дано ціле число n і масив дійсних чисел $X = \{x_i\}; i = \overline{1, n}$. Обчислити середнє арифметичне і середнє геометричне чисел масиву, використовуючи формули:

$$S := \frac{1}{n}\sum_{i=1}^{n} x_i ; \qquad P = \sqrt[n]{\prod_{i=1}^{n} x_i} .$$

```
Program OM_2;
Var
X: array [1. 100] of real;
n, i: integer;
S, P: real;
Begin
     writeln('Введіть розмір масиву n');
     read(n);
     writeln('Введіть масив X');
     for i := 1 to n do read(x[i]);
          S := 0;
          P := 1;
     for i := 1 to n do
          begin
          S := S + x[i];
          P := P * x[i];
          end;
     S := S/n;
     P := exp(1/n * ln(P));
     writeln('S =', S:6:2);
     writeln('P =', P:10:2);
End.
```

Приклад 3. Пошук мінімального і максимального елементів масиву.

Даний дійсний масив T = {t_i}, i = 1, 10. Поміняти місцями мінімальний і максимальний елементи масиву і вивести масив після обміну.

У цьому завданні додатково треба знати місцеположення мінімального і максимального елементів, тому одночасно з пошуком треба запам'ятовувати індекси. Введемо позначення:

min – мінімальний елемент;

imin – індекс мінімального елементу;

max – максимальний елемент;

imax – індекс максимального елементу.

```
Program OM_3;
```

```
Var
T: array [1. 10] of real;
i, imin, imax: integer;
min, max: real;
Begin
     writeln('Введіть масив T');
     for i:= 1 to 10 do read(t[i]);
          min:= +1E6;
          max:= -1E6;
     for i:= 1 to 10 do
          begin
          if t[i]<min then
               begin
               min:= t[i];
               imin:= i;
               end;
          if t[i]>max then
               begin
               max := t[i];
               imax := i;
               end;
          end;
     t[imin]:= max;
     t[imax]:= min;
     for i := 1 to 10 do write(t[i]:6:2);
     writeln;
End.
```

3.1.3. Двомірні масиви

Двомірні масиви в математиці представляються матрицею:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

або скорочено можна записати:

$$A = \{a_{ij}\}_{m \times n},$$

де m – число рядків;

n – число стовпців;

i, j – індекси (номери) поточних рядка і стовпця, на перетині яких знаходиться елемент a_{ij}.

Опис матриці в розділі VAR задається структурним типом вигляду:

```
<им'я масива>: ARRAY [m1..m2, n1..n2] OF <тип элементів>;
```

де m1..m2 – діапазон значень індексу i, що визначає число рядків;

n1..n2 – діапазон значень індексу j, що визначає число стовпців.

По опису матриці у внутрішній пам'яті комп'ютера виділяється область з (m*n) послідовних осередків, в які при роботі програми записуються значення елементів матриці. Наприклад, по опису:

```
Var
A: array [1..3, 1..5] of real;
```

у пам'яті комп'ютера виділяється область, що складається з 3*5=15 послідовних осередків, для запису елементів матриці речовинного типа. З опису видно, що матриця складається з трьох рядків і п'яти стовпців.

Звернення до окремих елементів матриці здійснюється за допомогою змінної з двома індексами, наприклад:

$a[i, j] \Rightarrow a_{ij}$;

$a[2,3] \Rightarrow a_{23}$;

$a[2*n, k+1] \Rightarrow a_{2n,k+1}$.

Для поелементного введення і виведення матриці використовується подвійний цикл ***FOR***. Якщо задати індекс *i* як параметр зовнішнього циклу, а індекс *j* як параметр внутрішнього циклу, то введення - виведення матриці здійснюється по рядках.

Приклад 1. Організувати введення цілочисельної матриці по рядках.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Опис матриці разом з поточними індексами має вигляд:

```
Var
M: array [1..2, 1..3] of integer;
i, j: integer;
```

Введення в програмі реалізується у формі діалогу, тобто супроводжується відповідним повідомленням:

```
Begin { почало програми}
writeln('Введіть матрицю M');
```

```
for i := 1 to 2 do
for j := 1 to 3 do read(m[i, j]);
```

На клавіатурі найнаочніше набирати елементи матриці по рядках.

1 2 3 <***Enter***>

4 5 6 <***Enter***>

Приклад 2. Організувати виведення матриці M на екран.

Виведення матриці треба реалізувати в зручному для читання вигляді, тобто щоб на одному рядку екрану розташовувався один рядок матриці. З цією метою в тіло зовнішнього циклу, крім внутрішнього, включається ще оператор ***WRITELN***, який переводить курсор до початку наступного рядка екрану після виведення поточного рядка матриці.

```
for i := 1 to 2 do
begin
for j := 1 to 3 do write(m[i, j]:3);
writeln
end;
```

Вид матриці на екрані буде наступним:

1 2 3

4 5 6.

3.1.4. Обробка матриць

Базовими алгоритмами обробки матриць є ті ж алгоритми, які використовуються при обробці одномірних масивів. Проте реалізацію цих алгоритмів можна умовно розглядати для двох типів завдань.

1. Алгоритми реалізуються при обробці всіх елементів матриці.
2. Алгоритми реалізуються усередині кожного рядка або кожного стовпця матриці.

Розглянемо реалізацію алгоритмів для завдань першого типа.

Приклад 1. Дана матриця дійсних чисел $A = \{a_{ij}\}_{4\times6}$. Обчислити значення $Z = \frac{P1}{|P2|}$, де P1 і P2 – добутки позитивних і негативних елементів матриці відповідно.

```
Program DM_1;
Var
A: array [1..4, 1..6] of  real;
i, j: integer;
P1, P2, Z: real;
Begin
     writeln('Введіть матрицю A');
     for i := 1 to 4 do
     for j := 1 to 6 do read(a[i,j]);
          P1 := 1;
          P2 := 1;
     for i := 1 to 4 do
     for j :=1 to 6 do
          begin
          if  a[i,j]>0 then P1 := P1 * a[i,j];
          if  a[i,j]<0 then P2 := P2 * a[i,j];
          end;
     Z := P1/abs(P2);
     writeln('Z='  Z:10:2);
End.
```

Приклад 2. У квадратній цілочисельній матриці $B = \{b_{ij}\}_{5\times5}$ обчислити модуль різниці між числом нульових елементів, що стоять нижче за головну діагональ, і числом нульових елементів, що стоять вище за головну діагональ.

Введемо позначення:

L1 – число нульових елементів нижче за головну діагональ;

L2 – число нульових елементів вище за головну діагональ;

L= |L1-L2|.

```
Program DM1_2;
Var
B: array [1..5, 1..5] of  integer;
i, j L1, L2, L: integer;
Begin
     writeln('Введіть матрицю B');
     for i := 1 to 5 do
     for j := 1 to 5 do read(b[i,j]);
          L1 := 0;
          L2 := 0;
```

```
     for i := 1 to 5 do
     for j := 1 to 5 do
     if b[i,j]=0 then
          begin
          if  i>j then L1 := L1+1;
          if  i<j then L2 := L2 +1;
          end;
     L:= abs(L1 – L2);
     writeln('L=', L);
End.
```

Тепер розглянемо реалізацію алгоритмів для завдань другого типу.

Приклад 3. У матриці $X = \{x_{ij}\}_{3\times 6}$ дійсних чисел перший елемент кожного рядка поміняти місцями з мінімальним елементом цього рядка. Вивести матрицю X після обміну.

```
Program DM2_1;
Var
X: array [1..3,1..6] of  real;
i, j, jmin: integer;
min: real;
Begin
     writeln('Введіть матрицю X');
     for i:= 1 to 3 do
     for j := 1 to 6 do read(x[i,j]);
     for i:=1 to 3 do
          begin
          min:=+1E6;
          for j:=1 to 6 do
          if x[i,j]<min then
               begin
               min:=x[i,j];
               jmin:=j;
               end;
               x[i,jmin]:=x[i,1];
               x[i,1]:=min;
          end;
     for i:=1 to 3 do
          begin
          for j:=1 to 6 do write (x[i,j]:6:1);
          writeln;
          end;
End.
```

Приклад 4.. Дана матриця дійсних чисел $C = \{c_{ij}\}_{8\times 4}$. Обчислити середнє арифметичне кожного стовпця. Результат оформити у вигляді одномірного масиву $S = \{S_j\}; \ j = 1, 4$.

```
Program DM2_2;
Var
C: array [1..8, 1..4] of  real;
S: array [1..4] of  real;
i, j: integer;
Begin
     writeln('Введіть матрицю C');
     for i := 1 to 8 do
     for j := 1 to 4 do read(s[i,j]);
     for j := 1 to 4 do
          begin
          s[ j ]:= 0;
          for i := 1 to 8 do s[ j ]:= s[ j ]+ s[i,j];
          s[ j ]:= s[ j ]/8;
          end;
     for j := 1 to 4 do write(s[ j ]:8:2)
     writeln;
End.
```

У даній програмі слід звернути увагу на те, що при обчисленні кожного елементу s[j] організований подвійний цикл, в якому індекс ***j*** є зовнішнім параметром циклу, а індекс ***i*** - внутрішнім. Це забезпечує обробку елементів матриці по стовпцях.

3.2. Рядки

Перейдемо тепер до вивчення масивів спеціального вигляду - лінійних масивів, що складаються тільки з символів – ***рядків***. Для обробки текстових даних використовується строковий тип даних String. ***Рядок*** –це послідовність символів з кодової таблиці, ув'язнена в апострофи. Кожен рядок характеризується своєю поточною довжиною (кількістю символів, наявних в рядку в даний момент), порядком розташування символів. За умовчанням для рядків визначена максимальна довжина, рівна 255 символам, але її можна змінити, вказавши при описі:

```
string [ число символів ].
```

Якщо рядок довше за максимальну довжину, то символи, що не помістилися, відкидаються. Символьним константам можна привласнювати імена.

```
Const st='строка';
```

Символьні змінні описуються в розділі опису змінних з описувачем ***String.***

```
Var  st1,st2:string[10];
st3:string;
     begin
     st1:='строка';
     end.
```

До будь-якого символу в рядку можна звернутися точно так, як і до елементу одномірного масиву, тобто вказавши ім'я рядка і індекс символу в цьому рядку.

Наприклад: **st[3]='р'.** При цьому для елементу рядка дозволені ті ж операції і функції, що і для типу **char**.

Рядки виводяться і вводяться за допомогою операторів ***READ, READLN, WRITE, WRITELN*** без організації циклів.

Над рядками виконується операція зчеплення, яка дозволяє з'єднати два або більш рядків в один без роздільників.

Приклад**: st1:='Кро';**

```
st2:='ил';
st3:=st1+'код'+st2;
```

Результат: **st3='Крокодил';**

Над рядками виконуються операції порівняння: =, <, >, <=, >=, <>. Рядки порівнюються посимвольно зліва направо до першого результату або до вичерпання символів рядку.

Наприклад: 'азбука'='азбука', оскільки всі символи поелементно співпадають.

Наприклад: 'школа'<'школьник'. Результат порівняння (true), оскільки

'ш'='ш', 'к'='к', 'о'='о', 'л'='л', 'а'<'ь' (символ 'а' розташований в кодовій таблиці раніше символу 'ь').

3.2.1. Функції обробки рядків

Зчеплення – ***concat*(строка1, строка2..)**, Аналогічна операції зчеплення.

Приклад*:* Початкові дані**:** `a='код', b='ил'.`

Оператор: `s:=concat('кро',a,b).`

Результат: `s='крокодил'.`

Копіювати – ***Copy* (рядок, число1, число2)**. З вказаного рядка виділяється підрядок, починаючи з позиції, заданої числом 1, завдовжки, заданої числом 2.

Приклад**:** Початкові дані: `s='крокодил'.`

Оператор: `b:=copy(s,2,3).`

Результат: `b='рок'.`

Позиція – ***Pos*(строка1, строка2**). Відшукує перше входження рядка 1 в рядок 2 і повертає номер початкової позиції входження або нуль, якщо строка1 не входить в рядок 2.

Приклад: Початкові дані: `s='крокодил'.`

Оператор: `i:=pos('око',s).`

Результат**:** `i=3.`

Оператор: `i:=pos('я','крокодил').`

Результат**:** `i=0.`

Довжина – ***length*(рядок)**. Повертає довжину рядка – аргументу.

Приклад: Початкові дані: `s='крокодил'.`

Оператор: `j:=length(s).`

Результат: `j=8.`

3.2.2. Процедури обробки рядків

Вставити – ***insert*(строка1, строка2, число)**. Вставляє рядок 1 в рядок 2, починаючи з позиції, заданої числом. Якщо в результаті виходить рядок довжини більше максимальної, то він усікається справа.

Приклад*:* Початкові дані: `S='крокодил'.`

Оператор: `d:=copy(s,3,3).`

Результат: `d='око'.`

Оператор: `insert('h',d,3).`

Результат: `d='окно'.`

Видалити – ***delete*(*рядок, число1, число2*)**. Видаляє з рядка підрядок, починаючи з позиції, заданої числом 1, довжиною, заданою числом 2. Якщо число 1 більше розміру рядка, то підрядок не видаляється. Якщо число 2 більше кількості, що була, то видаляються символи до кінця рядку.

Приклад: Початкові дані: **S='крокодил'.**

Оператор: **delete(s,4,3).**

Результат: ='**кроил'.**

Оператор: **delete(s,1,1).**

Результат: **s='роил'.**

Перетворити число в рядок – ***str*(*число[:M[:N]],строка*)**. Перетворить число в рядок. M задає загальну кількість символів, що одержуються в рядку, N – для дійсних чисел (типу real) задає кількість цифр в дробовій частині.

Приклад:

Оператор: **str(123,s).**

Результат: **s='123'.**

Перетворити рядок в число – ***val*(*рядок, число, код*)**. Перетворить рядок символів у внутрішні представлення числа. Код указує номер неправильного символу або рівний 0 у разі успішного перетворення.

Приклад:

Оператор: **val('+12.3',v,k).**

Результат: **v=12.3 k=0** { перетворення пройшло успішно}

Оператор: **val('23+5',v,k).**

Результат: **v=неправильно, k=3** { помилка при спробі перетворити третій символ}

3.3. Записи

Комбінований тип характеризує об'єкти, звані записами. ***Запис*** – це складна змінна з декількома компонентами. На відміну від масивів компоненти запису (поля) можуть мати різні типи, і доступ до них здійснюється не по індексу,

а на ім'я поля. При визначенні комбінованого типа задаються ім'я і тип кожного поля.

Тип має наступну структуру:

```
type   <им'я_типу> = RECORD <список_полей> END;
де     RECORD, END – зарезервовані слова (запис, кінець);
```

<им'я_типа> – правильний ідентифікатор;

<сп_ полів > – список полів; є послідовністю розділів запису, між якими ставиться крапка з комою.

До кожного компоненту запису можна звернутися, використовуючи ім'я змінної типу запису і ім'я поля, розділених крапкою.

Приклад:

```
type СотрФірм = RECORD
ФІБ: array [1..10] of char;
Оклад: integer;
Адреса: RECORD
Вулиця : array [1..10] of char;
Ндома,Нкв : integer;
END;
END;
var сотф : СотрФірм;
або
var сотф :RECORD
ФІБ: array [1..10] of char;
Оклад: integer;
Адреса: RECORD
Вулиця : array [1..10] of char;
Ндома,Нкв : integer;
END;
END;
Звернення:сотф.Оклад:=1344;
сотф.Адрес.Ндома:=12;
сотф.Адрес.Нкв:=34;
```

3.4. Множини

Множини – це набори однотипних логічно пов'язаних один з одним об'єктів. Кількість елементів, що входять в множину, може мінятися в межах від 0 до 256 (множина, що не містить елементів, називається порожньою). Саме непостійністю кількості своїх елементів множини відрізняються від масивів і записів.

Дві множини є еквівалентними тоді і тільки тоді, коли всі елементи однакові, причому порядок проходження елементів в множині байдужий.

Опис типу множини має вигляд:

<им'я типу>=**SET OF** <базовий тип>

де < им'я типу > – правильний ідентифікатор;

SET,OF – зарезервовані слова;

<базовий тип> – базовий тип елементів множини, може використовуватися будь-який порядковий тип, окрім ***word, longint, integer.***

Для завдання множини використовується так званий конструктор множини: список специфікацій елементів множини, відокремлюваних один від одного комами, список обрамляється квадратними дужками.

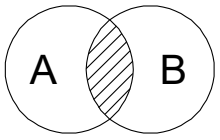
Операції над множинами:

Результатом операції ***перетину*** двох множин (A*B) буде множина C, що складається тільки з тих елементів які належать, як множині A, так і множині B.

Приклад: [1,2,3,4]*[3,4,5,6] результат [3,4]

C = A * B

C: = A * B

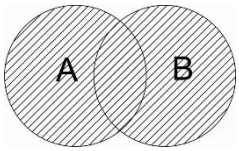


Результатом операції ***об'єднання*** безлічі A+B буде множина C, що включає як всі елементи множини A, так і всі елементи множини B.

Приклад: [1,2,3,4]+[3,4,5,6] результат [1,2,3,4,5,6]

C = A ∪ B

C: = A + B

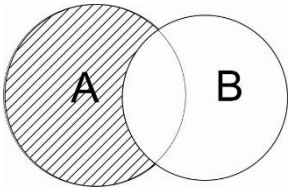


Результатом операції ***різниці*** двох безлічі A-B, буде множина C, що складається тільки з тих елементів множини A, які не входять в множину B.

Приклад: [1,2,3,4] – [3,4,5,6] результат [1,2]

C = A \ У

C: = A – B



Результатом операції ***порівняння*** A=B буде TRUE, а операції A< >B буде FALSE, тільки тоді, коли A і B містять одні і ті ж елементи.

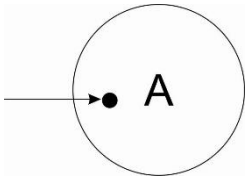
Результатом операції порівняння A<=B буде TRUE, якщо множина A є підмножиною множини B.

Результатом операції порівняння A>=B буде TRUE, якщо множина A включає всі елементи множини B.

Результатом операції ***приналежності*** X до A буде TRUE, якщо значення X будь якого порядкового типа T є елементом множини A того ж типу T.

X ∈ A

If X in A thin



Приклад операцій приналежності і об'єднання множин.

```
PROGRAM   Dem_Set_Type;
USES  Crt;
TYPE  SetType =set of char;
{ оголошення окремого типа для передачі параметрів в My
function}
VAR    Myset, Set1, Set2  :Settype;
           CH :char;
FUNCTION  Myfunction (Var set:  Set type) : char;
VAR  Flag: char;
BEGIN
       REPEAT
     Flag:=UpCase(ReadKey);
     UNTIL  Flag IN Var Set;
Writeln(' Правильно!!!');
     Myfunction := Flag;
END;
BEGIN
     MySet:= ['Y','N'];
     Writeln('Допускаються відповіді тільки Y або N');
     CH:= Myfunction (myset);
     Set1:=['K'];
     Myset := Set1 + Myset; { Злиття двох множин }
     Writeln('Допускається також K');
     CH :=Myfunction (Myset);
```

```
        REPEAT
        UNTIL  KeyPressed;  { Очікується натиснення клавіші для
повернення в початок}
    END.
```

3.5. Файли

У першому розділі ми вже розглядали введення інформації з клавіатури і виведення її на екран. Проте процес введення з консолі вельми трудомісткий, а результат висновку на консоль - недовговічний. На щастя, існує зручніший спосіб записувати, зберігати, пересилати і з потреби прочитувати інформацію з постійної пам'яті комп'ютера. Для цього застосовуються файли. ***Файл*** - це самостійна послідовність символів, записана в постійну пам'ять комп'ютера.

У англійській мові слово "file" має цілком зрозумілий сенс: "низка", що дуже добре відображає внутрішню структуру будь-якого файлу. Файл - це саме низка символів, причому зв'язаних в певній послідовності: символи файлу не можуть по своєму бажанню перестрибувати з одного місця на інше.

"Самостійність" файлів полягає в тому, що вони не залежать від роботи якої-небудь програми. І навіть якщо вимкнути комп'ютер, файли продовжуватимуть своє існування на вінчестері або на дискеті.

Файли можуть зберігати в собі все, що піддається кодуванню:

- початкові тексти програм або вхідні дані (тести);
- машинні коди виконуваних програм (ігри, віруси, повчальні і сервісні програми, ін.);
- інформацію про поточний стан якого-небудь процесу;
- різні документи, у тому числі і Інтернет -сторінки;
- картинки (малюнки, фотографії, відео);
- музику;
- і т.д. і т.п.

Переваги використання файлів наступні:

- Файли корисні, якщо об'єм вхідних даних перевершує посильний при ручному введенні. (Крайнім є випадок, коли вхідні або вихідні дані свідомо не можуть поміститися в оперативній пам'яті.)
- Файли потрібні, якщо доводиться багато разів вводити одну і ту ж інформацію, з мінімальними змінами або зовсім без змін (наприклад, при відладці програми).
- Файли необхідні, якщо потрібно зберігати інформацію про результати роботи програми, одержані при введенні різних вхідних даних (тобто: при пошуку помилок в програмі).

У мові Pascal є можливість роботи з трьома видами файлів:

- текстовими;
- що типізуються;
- що нетипізуються.

Останні два типи об'єднуються під назвою ***бінарні***: інформація в них записується по байтах і тому не доступна для перегляду або редагування в зручних для людини текстових редакторах, зате такі файли компактніші, ніж текстові.

На відміну від бінарних, ***текстові*** файли можливо створювати, переглядати і редагувати "уручну" - в будь-якому доступному текстовому редакторі. Крім того, при прочитуванні даних з текстового файлу немає необхідності піклуватися про їх перетворення: у мові Pascal є засоби автоматичного перекладу вмісту текстових файлів в потрібний тип і формат, і це дозволяє заощадити немало часу і сил.

3.6. Підпрограми

3.6.1. Структура складної програми

Будь-яка програма в Turbo Pascal може бути розбита на ряд самостійних програмних одиниць - підпрограм. Таке розділення викликане двома причинами.

1. Економія пам'яті. Кожна підпрограма записується в програмі один раз, тоді як звертатися до неї можна багато разів з різних точок програми.

2. Структуризація програми. Алгоритм рішення задачі може бути достатньо складним, тому доцільно виділити самостійні смислові частини алгоритму і оформити їх у вигляді підпрограм. Будь-яка підпрограма в свою чергу може містити підпрограми нижчого рівня. Такі структуровані програми легше зрозуміти, і вони зручні у відладці.

Надалі для простоти викладу розглянемо тільки такі підпрограми, які не містять усередині інших підпрограм. Розділ підпрограм включається в описову частину основної програми.

Структура складної програми має вигляд:

Program <ім'я основної програми>;

TYPE {розділ типів}
……
VAR {розділ опису змінних}
……{основної програми}
⟨Розділ опису підпрограм⟩ } Розділ опису основної програми

BEGIN {початок основної програми}
……
END. {кінець основної програми} } Розділ операторів основної програми

Розділ типів TYPE в основній програмі може бути відсутнім. Його призначення розглянемо пізніше.

Описи підпрограм розташовуються услід за розділом опису змінних основної програми VAR. Число підпрограм може бути довільним. У Turbo Pascal розрізняють два види підпрограм: процедури і функції. У окремих програмах можуть бути відсутніми або процедури, або функції.

3.6.2. Процедури

Будь-яка процедура складається із заголовка і тіла процедури. Тіло процедури оформляється за тими ж правилами, що і основна програма, тобто

складається з розділу опису змінних і розділу операторів. Проте закінчується тіло процедури символом **';'**.

Загальний вид опису процедури такий:

$$PROCEDURE < им'я > (< список\ формальних\ параметрів >);$$

$$\left.\begin{array}{l}\left.\begin{array}{l}VAR\\ \ldots\ldots\ldots\end{array}\right\} Розділ\ опису\ змінних\\ \left.\begin{array}{l}BEGIN\\ \ldots\ldots\ldots\\ END;\end{array}\right\} Розділ\ операторів\end{array}\right\} Тіло\ процедури$$

Список формальних параметрів служить для зв'язку процедури з основною програмою. У списку перераховуються вхідні і вихідні параметри з вказівкою їх типів. Вхідні параметри є початковими даними для процедури, а вихідні параметри визначають результати обчислень процедури, які передаються в основну програму.

У Турбо Паскалі допускається запис заголовка процедури без списку параметрів:

```
PROCEDURE <им'я>;    Звернення до процедури
```

Опис процедури сам по собі ніяких дій не викликає. Щоб виконати процедуру, в потрібній точці основної програми необхідно записати оператор виклику процедури.

```
<им'я процедури>(<список фактичних параметрів>);
```

Фактичні параметри замінюють формальні параметри при виконанні процедури. Фактичні параметри можуть бути константами, змінними або виразами.

Змінні, описані в основній програмі, є ***глобальними*.** Такі змінні можна використовувати в будь-якій точці програми, у тому числі і в процедурі. ***Локальні*** змінні визначаються в розділі опису VAR усередині процедури. Вони мають сенс тільки в процедурі і недоступні основній програмі.

Приклад програми з процедурою. Обчислити значення:

$$Z = \frac{a^5 + a^{-5}}{2a^7},$$

де ***a*** - задане дійсне число.

Введемо позначення:

$$r1 = a^5; \quad r2 = a^{-5} = \left(\frac{1}{a}\right)^5; \quad r3 = a^7.$$

У цьому завданні потрібно багато разів реалізувати алгоритм зведення в цілий ступінь. Доцільно використовувати процедуру, в якій даний алгоритм можна формально описати як алгоритм накопичення добутку.

$$P = x^n = \underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{n \text{ раз}} = \prod_{i=1}^{n} x,$$

де i - номер кроку обчислення (множення);

n - число кроків обчислень**.**

При описі процедури треба за допомогою списків параметрів пов'язати формальний параметр X з підставою ступеня, параметр n - з показником, а результат виконання процедури P- з фактичним результатом. Оскільки в завданні потрібно обчислити три операції зведення в ступінь ($r1, r2, r3$), то в основній програмі треба записати три оператори виклику процедури.

Програма має вигляд:

```
Program Primer_1;
Var
a r1, r2, r3, z: real;
Procedure ST (x: real; n: integer; var P: real);
var
i: integer;
begin
     P := 1;
     for i := 1 to n do P := P * x;
end;
Begin { початок основної програми}
     writeln('Введіть число a');
     read (a);
     ST(a, 5 r1);
     ST(1/a, 5 r2);
     ST(a, 7 r3);
     Z := (r1 + r2) / (2 * r3);
```

```
        writeln('Z='  Z:6:2);
End.
```

У заголовку процедури з ім'ям ST вказані два вхідні формальні параметри (x, n) і один вихідний (P).

Виконання програми завжди починається з операторів основної програми. В даному випадку після введення заданого числа (a) послідовно викликається три рази процедура ST. При кожному виклику відбувається відповідна заміна формальних параметрів (x, n*)* на фактичні, і обчислений результат через формальний параметр P привласнюється фактичним змінним r1, r2, r3 відповідно. Далі обчислюється значення Z, і результат виводиться на екран.

Формальні і ***фактичні*** параметри повинні бути узгоджені один з одним по кількості, типу і порядку проходження. Це означає, що кількість формальних параметрів повинна дорівнювати кількості фактичних параметрів, і кожен формальний параметр повинен мати той же тип і займати в списку те ж місце, що і відповідний йому фактичний параметр.

У списку формальних параметрів виділяється два види параметрів: параметри-значення і параметри-змінні. Механізм заміни для кожного виду параметрів різний.

Параметри-значення грають роль ***вхідних*** параметрів. Фактичним параметром, відповідним параметру-значенню, може бути константа, змінна або вираз. Параметри-значення є локальними змінними для процедури. Для них в пам'яті комп'ютера тимчасово виділяються осередки, в які передаються копії обчислених значень фактичних параметрів. При виконанні процедури параметри-значення можуть змінюватися**,** проте відповідні їм фактичні параметри залишаються без зміни.

Параметри–змінні є ***вихідними*** параметрами процедури. Перед параметрами-змінними в списку ставиться службове слово VAR. Як відповідні їм фактичні параметри можуть бути тільки змінні. При виклику процедури відбувається заміна імені параметра-змінної на ім'я фактичної змінної, тобто в процедуру

передається адреса фактичної змінної. Всі дії в процедурі виконуються безпосередньо над фактичним параметром, а не його копією. Тому будь-яка зміна формального параметра-змінної приводить до зміни відповідного йому фактичного параметра.

Розглянемо два приклади, що ілюструють поняття параметра-значення і параметра-змінної.

Приклад 1.

```
Program P1;
Var
x: integer;
Procedure Z (y: integer);
     begin
     y := 1;
     end;
Begin
     x := 0;
     Z(x);     виклик процедури}
     writeln('x=',  x);
End.
```

У описаній процедурі Z формальний параметр y є параметром-значенням, тому його зміна в процедурі (y := 1;) не впливає на значення фактичного параметра x. Після виконання програми на екран буде виведено: x = 0.

Приклад 2.

```
Program P2;
Var
x: integer;
Procedure Z (var y: integer);
begin
     y := 1;
end;
Begin
     x := 0;
     Z(x);
     writeln ('x=', x);
End.
```

У даній процедурі Z формальний параметр У є параметром-змінною, тому його значення після виконання процедури привласнюється фактичному параметру x. На екран буде виведено: x = 1.

3.6.3. Функції

Функція відрізняється від процедури тим, що результат її роботи повертається в основну програму у вигляді значення функції. Тому для функції необхідно вказати тип результату, а в тілі функції повинен бути присутнім оператор привласнення, в лівій частині якого записується ім'я функції.

Загальний вид опису функції:

```
FUNCTION<им'я> [(<список фактичних. параметрів>)]: <тип>,
VAR
. . .
BEGIN
<им'я функції > := <результат>;
END;
```

Оскільки результат виконання функції повертається в основну програму через ім'я функції, теозвернення до функції можна записати, аналогічно стандартним функціям, у вигляді операнда у виразі:

```
<им'я функції> [(< список фактичних. параметрів>)]:
```

При обчисленні виразу операнд звернення до функції замінюється значенням функції.

Зв'язок формальних параметрів з фактичними параметрами здійснюється за тими ж правилами, що і в процедурі. Різниця полягає лише в тому, що імені функції може бути привласнено тільки одне значення. Якщо у функції обчислюються декілька вихідних змінних, то вони можуть повертатися в основну програму через списки параметрів.

Для функції, аналогічно процедурі, справедливі поняття глобальних і локальних змінних.

Приклад програми з функцією. Обчислити значення:

$$Z = \frac{a^5 + a^{-5}}{2a^7},$$

де **a** - задане дійсне число, користуючись підпрограмой- функцією.

```
Program Primer_2;
Var
a, Z: real;
Function ST (x: real; n: integer): real;
var
```

```
i: integer;
P: real;
begin
     P := 1;
     for i := 1 to n do P := P * x;
     ST := P;
end;
Begin { почало основної програми}
     writeln('Введіть число a');
     read(a);
     Z := (ST(a, 5)+ ST(1/a, 5)) / (2 * ST(a, 7));
     writeln('Z='  Z:6:2);
End.
```

У даній програмі звернення до функції здійснюється за допомогою трьох операндів, записаних у виразі для обчислення Z.

3.7. Модулі мови Pascal

Модуль - це частка програми, компільована окремо від решти її частин. Саме можливість роздільної компіляції і є основною перевагою модулів.

Проста модульність програми може досягатися за рахунок застосування процедур і функцій, проте цього не завжди достатньо. Якщо всі підпрограми містяться в одному файлі, то виправлення єдиної помилки в якій-небудь підпрограмі приведе до неминучої перекомпіляції всього коду. А при сучасних розмірах програм компіляція може тривати навіть не хвилини, а годинник.

Крім того, якщо колектив програмістів пише одну велику програму (а саме в таких умовах працюють сьогодні всі виробники програмного забезпечення), то кожному з них потрібно забезпечити більш менш незалежний фронт робіт. У такій ситуації модулі, які зберігаються кожен в окремому файлі і можуть бути відредаговані, відкомпільовані і протестовані незалежно від решти частин програми, є якнайкращим рішенням цієї проблеми.

Декілька модулів, що є складовими частинами однієї програми, об'єднуються в ***бібліотеку***. Наприклад, разом з компілятором мови Pascal поставляються стандартні бібліотеки, що містять найважливіші підпрограми обробки даних. Перерахуємо найпоширеніші модулі, що входять до складу стандартних

бібліотек мови Pascal. Докладний опис цих бібліотек можна знайти в будь-якому довідковому виданні.

Модуль ***System*** є основним: у ньому містяться всі вивчені нами стандартні процедури і функції обробки арифметичних виразів, множин, рядків і т.п. Спеціального підключення цей модуль не вимагає: його вмістом можна користуватися за умовчанням.

Нагадаємо, що цей модуль містить наступні типи підпрограм:

Модуль ***Crt*** служить для організації "якісного" висновку на екран. Модуль ***WinCrt*** призначений для створення програм, що підтримують простий віконний інтерфейс.

Модуль ***Printer*** дозволяє проводити виведення інформації не на консоль, а на принтер (під операційною системою DOS).

Модуль ***WinPrn*** є аналогом модуля ***Printer*** для операційної системи Windows.

Модуль ***Dos*** дозволяє обмінюватися інформацією з операційною системою. Системний час, переривання, стани параметрів оточення, процедури обробки процесів, робота з дисковим простором - всім цим займається модуль `Dos`.

Модуль ***WinDos*** є аналогом модуля ***Dos*** для операційної системи Windows.

Модуль ***Strings*** дозволяє перейти від стандартних рядків мови Pascal до рядків, обмеженим нулем. На відміну від звичайних рядків, чия довжина не може перевищувати 255 символів, ці рядки можуть складатися з 65 535 символів, причому кінець кожного такого рядка помічений символом `#0`.

Модуль ***Graph*** містить різноманітні підпрограми, які дозволяють створювати на екрані різні малюнки з багатоколірних геометричних фігур. Модуль управляє також палітрами, фактурами фону і шрифтами.

Модуль ***Overlay*** надає можливість робити великі програми оверлейними (що багато разів використовують одну і ту ж область пам'яті).

Модуль ***WinApi*** відповідає за створення динамічних бібліотек. Цей модуль притаманний лише новим версіям мови Pascal (наприклад, Turbo Pascal 7.0).

Для того, щоб підключити до програми який-небудь модуль, необхідно відразу після заголовка програми помістити наступний рядок:

```
uses <им'я_модуля>;
```

Якщо модулів, що підключаються, декілька, цей рядок прийме вигляд:

```
uses <им'я_модуля_1>,...,<им'я_модуля_N>;
```

Втім, абсолютно не обов'язково указувати імена всіх модулів, що так чи інакше фігурують в програмі. Досить вказати імена лише тих, до яких вона звертатиметься безпосередньо. А до кожного модуля, підключеного до головної програми, у разі потреби можна підключити інші модулі - і т.д.

Всі константи, типи даних, змінні, процедури і функції, описані в якому-небудь модулі, після його підключення до основної програми (або до іншого модуля) стають доступними цій програмі (або модулю) без додаткових оголошень.

3.7.1. Структура модуля

До складу модуля входять чотири секції (будь-яка з них може бути порожньою, але її заголовок все одно зобов'язаний бути присутнім):

```
unit <имя_модуля>;
interface     {секція зовнішніх зв'язків}
implementation      { секція реалізацій}
begin   {секція ініціалізації}
end.
```

Як приклад ми приведемо модуль, що містить арифметичні функції `min` і `max` для цілих чисел, а також функцію зведення в ступінь. Всі ці функції відсутні в стандартному модулі ***System***.

```
unit my_arifm;
   interface
       function min(a,b: longint): longint;
       function max(a,b: longint): longint;
       function deg(a,b: double): double;

    implementation
       function min;
```

```
          begin if a>b then min:= b
                        else min:= a;
          end;

       function max;
          begin if a<b then max:= b
                        else max:= a;
          end;

       function deg;
          begin deg:= exp(b*ln(a))
         end;
end.
```

Тепер, підключивши цей модуль до будь-якої своїй програмі, ви зможете користуватися цими трьома функціями. Решту функцій, необхідних в роботі підпрограм (наприклад, тригонометричні функції tg, ctg, sec, arcsin, arccos, arctg, arcctg, arcsec) користувач може додати в цей модуль самостійно.

Аналогічним чином, корисно одного разу написати і відладити підпрограми, оброблювальні динамічні структури даних (списки, дерева, стеки і т.п.), зберегти їх в спеціальному модулі, а потім користуватися раніше виконаною роботою знов і знов, не витрачаючи часу на повторне програмування.

3.7.2. Процедури модуля Crt

Раніше згадувалося, що модуль ***Crt***, що входить до складу стандартних бібліотек мови Pascal, містить засоби для роботи з екраном в текстовому режимі. Для того, щоб зробити працездатними всі описані нижче процедури і функції, ваша програма повинна підключити стандартний модуль ***Crt***:

```
uses crt;
```

Процедура ***Window(x1,y1,x2,y2: byte)*** створить на екрані вікно з координатами лівого верхнього кута в крапці (x1,y1) і координатами правого нижнього кута в крапці (x2,y2). Тепер активна область екрану буде обмежена цим вікном. Поточні координати курсора відлічуватимуться не від лівого верхнього кута екрану, а від лівого верхнього кута цього вікна.

Процедура ***ClrScr*** очистить весь екран (або активне вікно); курсор буде поміщений у верхній лівий його кут.

Процедура ***ClrEol*** очистить поточний рядок, починаючи з поточної позиції курсора і до правого краю екрану (вікна).

Процедура ***DelLine*** видалить рядок, в якому знаходиться курсор.

Процедура ***InsLine*** очистить поточний рядок цілком. Курсор залишиться на колишній позиції.

Процедура ***TextBackground(color: byte)*** встановить колір фону.

Процедура ***TextColor(color: byte)*** встановить колір тексту, що виводиться.

Процедура ***Sound(hz: word)*** включить звуковий сигнал з частотою `hz` герц.

Процедура ***NoSound*** вимкне звуковий сигнал.

Процедура ***GotoXY(x,y: byte)*** перемістить курсор в задану позицію в межах поточного вікна (екрану).

Функція ***WhereX: byte*** обчислить положення курсора в поточному вікні (або на екрані): його горизонтальну складову. Нагадаємо, що координата `X` відлічується від лівого краю екрану (вікна).

Функція ***WhereY: byte*** обчислить положення курсора в поточному вікні (або на екрані): його вертикальну складову. Нагадаємо, що координата `Y` відлічується від верхнього краю екрану (вікна).

Процедура ***Delay(ms: word)*** припинить виконання програми на `ms` мілісекунд.

Функція ***KeyPressed: boolean*** відстежує натиснення клавіш (на клавіатурі).

Функція ***ReadKey: char*** повертає код символу, чия клавіша (або комбінація клавіш) була натиснута.

3.8. Практичні роботи до розділу

Практична робота № 6. Робота з масивами чисел

1. Пошукові завдання для масивів

Часто в програмуванні виникає завдання відшукати перший елемент, співпадаючий із заданим x. Для вирішення цього завдання можуть бути запропоновані наступні варіанти:

- лінійний пошук;
- пошук з бар'єром.

Особливістю ***лінійного пошуку*** є саме причини припинення пошуку:

- елемент знайдений (це програмується за допомогою логічної змінної)
- проглянуті всі елементи, але заданий елемент так і не знайшли (i > n)

Приклад:

```
Program UPR7-1
const n = 20; { кількість елементів в масиві}
var a: array [1..n] of integer;   { початковий масив}
 i, x: integer;
 f: boolean;
begin
 write ('задайте шуканий елемент');
 readln (x);
 writeln ('задайте елементи масиву');
 for i: = 1 to n do
 readln (a[i]);
 f: = false;   { елемент ще не знайдений}
 i: = 1;
 while (i < = n) and not f do
 if a[i]= x then f: = true    { знайшли}
 else i: = i + 1          { переходимо до наступного елементу}
 if f then writeln (' знайшли елемент з номером', i)
 else writeln ('такого елементу немає')
End.
```

Застосовується широко поширений прийом ***пошуку з бар'єром*****,** розташованим в кінці масиву. Використання бар'єру дозволяє спростити умову закінчення циклу, оскільки наперед ясно, що хоч би один елемент, рівний ***a***, в масиві є. При цьому необхідно збільшити розмір масиву на 1.

Приклад*:*

```
Program UPR7-2
const n = 20; { кількість елементів в масиві}
var a: array [1..n + 1] of integer; { початковий масив}
 i, x: integer;
begin
 write ('задайте шуканий елемент');
 readln (x); writeln ('задайте елементи масиву');
 for i: = 1 to n do
 readln (a[i]);
 a[n + 1]: = x;  { встановлений бар'єр, рівний x, в кінці}
 i: = 1;              { масиву}
 while a[i]<> x do
 i: = i + 1;    { переходимо до наступного елементу}
 if i <> n + 1 then writeln ('знайшли елемент з номером', i)
 else writeln ('такого елементу немає'):
End.
```

2. Сортування масивів

При сортуванні вставкою масив розділяється на дві частини: відсортовану і невідсортовану. Елементи з невідсортованої частини по черзі вибираються і вставляються у відсортовану частину так, щоб не порушити в ній впорядкованість елементів.

На початку роботи алгоритму відсортованою частиною масиву приймається тільки один перший елемент, а як невідсортована частина – це вся решта елементів. Таким чином, алгоритм складатиметься з (n – 1)-го проходу (n – розмірність масиву), кожний з яких включатиме чотири дії:

1. узяття чергового i-го невідсортованого елементу і збереження його додаткової змінної.
2. пошук позиції j у відсортованій частині масиву, в якій присутність узятого елементу не порушить впорядкованості елементів.
3. зрушення елементів масиву від i – 1-го до j-го управо, щоб звільнити знайдену позицію вставки.
4. вставка узятого елементу в знайдену j-ю позицію.

Програма, що реалізовує розглянутий алгоритм, матиме наступний вигляд:

```
   Program UPR7-3;
 Uses Crt;
```

```
  Const
           n =10; {довжина масиву}
  type
           TVector  = array  [1.n]  of  Real;
  Var
           Vector  :  TVector;
            B          :   Real;
            i,  j, K :   Integer;
  Begin
            ClrScr;
            Writeln (‘Введіть елементи масиву:’);
            For  i:=1 to n do Read (Vector i]); Readln;
  {- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
            for   i:=2 to n do
  begin
            B:=Vector [i]; { узяття невідсортованого елементу}
                { цикл пошуку позиції вставки}
            j:=1
            While (BVector j] )  do
            j:= j+1 { після закінчення циклу індекс j фіксує позицію}
                                        { вставка}
           { цикл зрушення елементу для звільнення позиції  вставки}
           for K:= i-1   downto j do
           Vector  [K+1]: = Vector[  [K];
            { Вставка узятого елементу на  знайдену позицію}
            Vector j]: = B;
             End;
  { - - - - - -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
         Writein (‘Відсортований  масив:’);
          For i: = 1 to n do write    (‘vector [i] : 8 : 2);
          Writeln;
  End.
```

При сортуванні ***вибором*** знаходять в масиві елемент з мінімальним значенням на інтервалі від 1-го елементу до n-го (останнього) елементу і міняють його місцями з першим елементом. На другому кроці знаходять елемент з мінімальним значенням на інтервалі від 2-го до n-го елементу і міняють його місцями з другим елементом. І так далі для всіх елементів до n-1-го.

Програма що реалізовує метод вибору, має наступний вигляд:

```
Program UPR7-4;
 Uses Crt;
 const
          n=20;  { довжина масиву}
 typc
          TVector = array 1.n of Real;
```

```
Var
         Vector : TVector;
         Min     :  Real;
         Imin, S: Integer;
         i          : Integer;
Begin
         Clr Srt:
        Writeln ('введіть елементи масиву:');
for i: = 1 to n do Read (Vector i]);  Readln;
{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
for S:=1 to n-1  do
begin
{ пошук мінімального елементу в  діапазоні від }
      {S-го елементу до n-го}
Min: = Vector S];
I min: = S;
For i: = S+1 to n do
             If   Vector  [i]  Min  then
             begin
             Min: = Vector i];
             I min: = I;
End;
 { обмін місцями мінімального і  S-го елементу}
     Vector Imin]: = Vector S];
      Vector S]: = Min;
       End;
 { - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
     Writeln ('Відсортований масив:');
     For i: = 1 to n do Write (Vector i]: 8 : 2);
     Writeln;
End.
```

При сортуванні ***обміном*** («бульбашкове сортування») зліва направо по черзі порівнюються два сусідні елементи, і якщо їх взаєморозташування не відповідає заданій умові впорядкованості, то вони міняються. Далі беруться два наступні сусідні елементи і так далі до кінця масиву. Після одного такого проходу на останньому n-ій позиції масиву стоятиме максимальний елемент («спливла перша бульбашка»). Оскільки максимальний елемент вже стоїть на своїй останньої позиції, то другий прохід обмінів виконується до n-1 елементу. І так далі. Всього потрібний (n-1) прохід. Програма, що реалізує метод, матиме наступний вигляд:

```
Program UPR7-5;
 Uses  Crt;
 Const
```

```
          N = 20;     { довжина масиву}
Type
          TVector = array 1.n of Real;
Var
Vector : Tvector;
            B         :  Real;
            i, K      : Integer;
begin
           ClrScr;
           Writeln (‘введіть елементи масиву:’);
           for i: = 1 to n do Read (Vector [i]); Readln;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
           for K: = n  downto  2 do
{ “спливання” чергового максимального елементу}
{ на К-ю позицію}
            for i: = 1 to K-1 do
            if  Vector [i] > Vector [i+1] then
 begin
            B: = Vector [i];
            Vector [i]: = Vector [i+1];
            Vector [i+1]: = B;
            End;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
            Writeln  (‘відсортований масив:’);
            For i:  = 1 to n  do Write (‘Vector [i]: 8 : 2’);
            Writeln;
End.
```

3. Сортування двомірних масивів

Неодхідно відсортувати елементи двомірного масиву по елементах другого рядка.

Початковий масив.	Результат
1 2 3 4 5	5 2 4 3 1
9 3 7 3 1	1 3 3 7 9
6 7 8 9 1	1 7 9 8 6

Від сортування одномірного масиву цей випадок відрізняється тільки тим, що переставляти потрібно не два порівнювані елементи, а два стовпці:

```
Program UPR7-6
 for i: = 1  to n – 1  do
 for j: = i+1 to n do
 if a [2, i] > a [2, j]
 then  for до: = 1 to n do
 begin r: = a [до, i]; a [до, i]; = a[до, j]; a [до, j]: =r
 end.
```

Приведемо приклад програми по сортуванню двомірного масиву методом вставки:

```
  Program UPR7-7;
uses crt;
const n=4;
var mas:array[1..n,1..n]of integer;
    buf,l,i,j,a,c,nextmas:integer;
    quit:boolean;
procedure print;
begin
 for i:=1 to n do
  begin
  for j:=1 to n do
   write(' ',mas'[i,j] ');
  writeln;
  end;
 writeln;writeln(Натисніть Enter для продовження !');
 readln;
end;
procedure next;
begin
 j:=j+1;
 if j=n+1 then begin j:=1;i:=i+1;end;
end;
procedure last;
begin
 j:=j-1;
 if j=0 then begin j:=n;i:=i-1;end;
end;
begin
 clrscr;
 for i:=1 to n do
  for j:=1 to n do
   mas[i,j]:=round(random*(9));
 print;
 i:=1;j:=1;
 for l:=2 to (n*n) do
 begin
  next;buf:=mas[i,j];a:=i;c:=j;
  last;quit:=true;
  while (buf<mas[i,j]) and(quit) do
  begin
   nextmas:=mas[i,j];
   mas[i,j]:=buf;
   next;mas[i,j]:=nextmas;last;last;
   if j=0 then quit:=false;
  end;
  i:=a;j:=c;
 end;
 print; end.
```

4. Рішення лінійних рівнянь за допомогою матриць

Завдання 1: **с**кладіть математичну модель наступної задачі та розв'яжіть її матричним способом.

Для виробництва препаратів **A, B** та **C** використовуються компоненти (сировина) **1, 2** та **3**. Запас цих компонентів в тоннах складає Z_1**,** Z_2 і Z_3 відповідно. Для виготовлення однієї таблетки препарату **A** необхідно a_1 мг компоненту **1**, a_2 мг компоненту **2** й a_3 мг компоненту **3**. Аналогічні величини відомі для препаратів **B** та **C:** b_1, b_2, b_3 й c_1, c_2, c_3. Необхідно визначити, скільки таблеток кожного препарату можна виготовити, якщо використати усі запаси сировини, що є у наявності. Значення параметрів a_i, b_i, c_i та Z_i (i=1, 2, 3) наведені в таблиці 1.

Таблиця 3.1 – Вихідні данні для завдання 1

	Витрати сировини (мг) на одну таблетку препарату			Запаси сировини (т)
	Препарат **A**	Препарат **B**	Препарат **C**	
Сировина **1**	1	2	3	9
Сировина **2**	3	2	1	12
Сировина **3**	2	3	1	9

Складання математичної моделі задачі почнемо з введення невідомих, які треба визначити. Позначимо x – кількість таблеток препарату **A**, які можна виготовити з сировини, що є у наявності, y – кількість таблеток препарату **B** та z – кількість таблеток препарату **C**. Тоді кількість мг сировини **1**, що використовується для виготовлення x таблеток препарату **A**, y таблеток препарату **B** та z таблеток препарату **C** може бути підрахована як сума доданків:

$$\boldsymbol{a}_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1 \cdot \boldsymbol{y} + \boldsymbol{c}_1 \cdot \boldsymbol{z} .$$

Аналогічно, кількість мг сировини **2**, що використовується для виготовлення x таблеток препарату **A**, y таблеток препарату **B** та z таблеток препарату **C** дорівнює

$$a_2 \cdot x + b_2 \cdot y + c_2 \cdot z;$$

а також кількість мг сировини **3**, що використовується для виготовлення x таблеток препарату **A**, y таблеток препарату **B** та z таблеток препарату **C** дорівнює

$$a_3 \cdot x + b_3 \cdot y + c_3 \cdot z.$$

Виходячи з умови, що використані усі запаси компонентів **1**, **2** та **3**, що складають Z_1, Z_2 та Z_3 тонн відповідно, можемо прирівняти $a_1 \cdot x + b_1 \cdot y + c_1 \cdot z = Z_1 \cdot 10^6$; $a_2 \cdot x + b_2 \cdot y + c_2 \cdot z = Z_2 \cdot 10^6$ та $a_3 \cdot x + b_3 \cdot y + c_3 \cdot z = Z_3 \cdot 10^6$.

Таким чином, підставляючи конкретні значення параметрів a_i, b_i, c_i та Z_i з таблиці 1, отримуємо математичну модель нашої задачі у вигляді системи лінійних рівнянь:

$$\begin{cases} x + 2 \cdot y + 3 \cdot z = 9 \cdot 10^6 \\ 3 \cdot x + 2 \cdot y + z = 12 \cdot 10^6 \\ 2 \cdot x + 3 \cdot y + z = 9 \cdot 10^6 \end{cases}.$$

Сутність матричного методу міститься в тому, що система лінійних рівнянь записується у вигляді: A·s = b, де A – матриця системи, b – вектор-стовпчик правих частин рівнянь, s – вектор невідомих змінних (в нашому випадку це вектор-стовпчик із компонентами x, y, z). Тоді розв'язок системи знаходиться за формулою: $s = A^{-1} \cdot b$.

$$A := \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 3 & 1 \end{pmatrix} \qquad b := \begin{pmatrix} 9 \\ 12 \\ 9 \end{pmatrix} \cdot 10^6$$

$$s := A^{-1} \cdot b \rightarrow \begin{pmatrix} 3250000 \\ 250000 \\ 1750000 \end{pmatrix}$$

Таким чином, отримуємо відповідь на завдання: використавши усі запаси трьох видів сировини можна виготовити 3 250 000 таблеток препарату **A**, 250 000 таблеток препарату **B** та 1 750 000 таблеток препарату **C**.

5. Завдання для самостійної роботи

1. Визначте витрати меланжу (суміш H_2O – 5%, HNO_3 – 85% та H_2SO_4 – 10%), олеуму (H_2SO_4 – 100%) і відробки сірчаної кислоти (суміш H_2O – 30% і H_2SO_4 – 70%) для приготування 1,2 кг нітруючої суміші з наступними параметрами: H_2O – 25%, HNO_3 – 15%, H_2SO_4 – 60%.

2. Аптека №2 закуповує бальзам «Біттнера» у трьох постачальників: у головного аптечного складу по 5,50 грн. за флакон, у ВАТ «Здоров'я-М» по 4,95 грн. за флакон, й у фірми «Фармас'ютикалз Ltd.» по 5,70 грн. за флакон. В попередньому місяці аптека зверталася до всіх трьох постачальників й витратила на закупку бальзаму «Біттнера» 4,095 тис. грн.; однак, якби вона не робила закупки на головному складі, витрачено було б всього 3,27 тис. грн. Відомо також, що якби всі три фірми продавали препарат по 5 грн., то на закупку тієї ж самої кількості флаконів витрачено було б 3,75 тис. грн. Скільки флаконів бальзаму «Біттнера» було закуплено в кожного з постачальників?

3. Для виготовлення лікарських препаратів P1, P2, P3 та P4 використовується сировина чотирьох типів S1, S2, S3 та S4. Дані про запаси сировини на фабриці

й витратам сировини на одну таблетку кожного препарату наведені в таблиці 3.2. Визначте, яку кількість таблеток Р1, Р2, Р3 та Р4 можна випустити, використавши усю сировину, що є в наявності.

Таблиця 3.2 – Вихідні данні для завдання 3

Тип сировини	Запас сировини, кг	Витрати сировини у мг на 1 таблетку препарату			
		P_1	P_2	P_3	P_4
S_1	49,033	20,3	2,8	35	5,4
S_2	47,562	12,2	24,8	3,5	14,7
S_3	35,886	2,4	15,3	12,8	20,5
S_4	50,883	8,5	5,8	28,4	32,3

4. На підприємство, де працюють співробітники чотирьох категорій привезли заробітну платню в купюрах такого номіналу: по 100 гривень – 1850 купюр, по 50 гривень – 230 купюр, по 10 гривень – 250 купюр, по 1 гривні – 740 купюр. Заробітна платня робітника 1-ї категорії складає 962 грн., 2-ї категорії – 713 грн., 3-ї категорії – 452 грн., 4-ї категорії – 261 грн. Визначте, скільки співробітників кожної категорії працює на підприємстві, якщо кожному співробітнику видали зарплату мінімальною кількістю купюр. Розподіл купюр для заробітної платні кожної категорії наведено в таблиці 3.3.

Таблиця 3.3 – Вихідні данні до завдання 4

Номінал купюр	Загальна кількість купюр	Розподіл купюр за категоріями			
		1 кат.	2 кат.	3 кат.	4 кат.
100 грн.	1850	9	7	4	2
50 грн.	230	1	0	1	1
10 грн.	250	1	1	0	1
1 грн.	740	2	3	2	1

5. В одному з цехів на хіміко-фармацевтичному виробництві працюють співробітники наступних спеціальностей: хіміки-технологи, оператори верстатів із ЧПУ, інженери хімічної промисловості та робітники. Відомо, що зарплата технолога складає 113 у.о., оператора верстатів із ЧПУ – 167 у.о., інженера – 3080 у.о., робітника – 99 у.о. Щоб видати зарплату співробітникам цеху привезли купюри номіналом по 50, 10, 5 та 1 у.о., причому таким чином, щоб кожному співробітнику видати заробітну платню мінімальною кількістю купюр. Купюр номіналом по 50 у.о. було 97 штук, купюр номіналом по 10 у.о. – 122, купюр номіналом по 5 у.о. – 38, й купюр номіналом по 1 у.о. – 165 штук. Визначте кількість хіміків-технологів, операторів верстатів із ЧПУ, інженерів хімічної промисловості і робітників у цеху.

6. Покупець придбав від кашлю 1 упаковку «Льодяників Доктора Тайса» і 1 упаковку льодяників «Доктор Мом» й заплатив за покупку 14 гривень 50 копійок. Однак за ту ж саму суму він міг би придбати 3 упаковки льодяників «Доктор Мом» й 2 упаковки таблеток «Пектусин», або, за ту ж саму суму, – 29 упаковок таблеток «Пектусин». Яка вартість кожного зі згаданих в задачі препаратів?

7. В таблиці 3.4 наведена поживна та енергетична цінність чотирьох продуктів, в таблиці 6 – добова потреба людини у поживних речовинах й енерговитрати в залежності від віку та роду занять. Визначте, якою кількістю даних у таблиці 5 продуктів можна забезпечити добові потреби дітей до 7 років у білках, жирах, вуглеводах і компенсувати енерговитрати.

Таблиця 3.4 – Вихідні данні для завдання 7

Продукти	У 100 г продукту			
	Білки, г	Жири, г	Вуглеводи, г	Енергетична цінність, ккал
Хліб пшеничний	7,0	0	55	230
Молоко	5,0	2,5	4,8	60
Масло вершкове	1,0	97	0	780
Картопля	2,0	0,1	19,7	90

8. В таблиці 3.4 наведена поживна та енергетична цінність чотирьох продуктів, в таблиці 3.5 – добова потреба людини у поживних речовинах й енерговитрати в залежності від віку та роду занять. Визначте, якою кількістю даних у таблиці 3.4 продуктів можна забезпечити добові потреби студентів у білках, жирах, вуглеводах і компенсувати енерговитрати.

Таблиця 3.5 – Вихідні дання для завдання 8

Вік та рід занять	Добові потреби			
	Білки, г	Жири, г	Вуглеводи, г	Енергови-трати, ккал
Діти до 7 років	70	65	250	1800
Студенти	105	100	415	2900
Дорослі	132	140	635	4100

9. В аптеці «36,6» покупець витратив 8,80 грн., придбавши 2 упаковки аспірину, 1 пачку чаю «Нефрофіт» і 1 флакон сиропу від кашлю «Пертуссін». Відомо, що в аптеці за рогом чай дешевше на 10%, а сироп дешевше на третину, й уся покупка обійшлась б у 7,44 грн. Однак, в центральній аптеці аспірин дорожче на 25 копійок, які складають третю частину його вартості в аптеці «36,6». Яка ціна за упаковку (флакон) кожного із згаданих препаратів в аптеці «36,6»?

10. Підприємством НВО «Фарм» використовується щодобово чотири види однакової сировини для виробництва чотирьох найменувань лікарських препаратів. Витрати сировини (мг) на одну упаковку кожного препарату наведені в таблиці 3.6. Щодобово витрачається сировини I – 240 г, сировини II – 250 г, сировини III – 340 г, сировини IV – 420 г. Визначте, скільки упаковок кожного препарату виробляється кожної доби.

11. Виробнича фірм «Доктор—А», що випускає п’ять найменувань лікарських препаратів, працює увесь рік в режимі п’ятидобового робочого тижня. В таблиці 3.7 вказана виробнича потужність підприємства (упаковок за день). Всього в

минулому році було випущено 124,4 тис. упаковок амідопірину, 47,1 тис. упаковок барбамілу, 161,2 тис. упаковок стоптуссіну, 146,9 тис. упаковок дриксенолу й 95,2 тис. упаковок фуросеміду. Визначте, скільки в минулому році було робочих понеділків, вівторків, серед, четвергів та п'ятниць.

Таблиця 3.6 – Вихідні данні для завдання 10

Препарати	Види сировини			
	I	II	III	IV
Аспаркам	2	3	4	5
Бісептол	1	2	5	6
Валідол	7	2	3	2
Цитрамон	4	5	6	8

Таблиця 3.7 – Вихідні данні для завдання 11

Лікарські препарати	День тижня				
	Пн	Вт	Ср	Чт	Пт
Амідопірин	400	500	300	600	700
Барбаміл	-	200	400	300	-
Стоптуссін	800	1500	-	400	600
Дриксенол	300	1000	700	500	400
Фуросемід	700	300	500	200	300

12. Аптека «Таблетки на Блюхера» працює п'ять днів на тиждень. В таблиці 3.8 наведено середнєдобову кількість продажів п'яти препаратів аптекою за останній рік. Всього в минулому році було 53 робочих тижня, за які було продано 41,9 тис. упаковок анальгіну, 41,8 тис. упаковок сустаку, 77,2 тис. упаковок простаплану, 47,2 тис. упаковок димедролу й 62,2 тис. упаковок но-шпа. Визначте, скільки свят в минулому році випало на понеділок, вівторок, середу, четвер і п'ятницю.

Таблиця 3.8 – Вихідні данні для завдання

Лікарські препарати	Середня кількість продажів (100 упаковок в день)				
	Пн	Вт	Ср	Чт	Пт
Анальгін	1	1	3	-	3
Сустак	-	1	4	3	-
Простаплан	3	5	-	4	3
Димедрол	1	1	5	-	2
Но-шпа	2	3	2	2	3

Практична робота № 7. Обробка рядків

Завдання 1. Заданий рядок, що складається із слів, розділених одним або декількома пропусками. Видалити повторні входження кожного слова.

Виділяємо слова, переписуємо їх в перший рядок двовимірного масиву, в другу записуємо ‘0’ для унікального слова і ‘1’- для слова, що повторюється. Потім формуємо рядок, що складається з елементів першого рядка масиву, у яких в другому рядку записано '0‘ і роздруковуємо рядок.

Текст програми:

```
program UPR8-1;
const nn=10;
type mas=array [1..2,1..nn] of string;
var a:mas;
      n:integer;
      s,ss:string;     { початковий і допоміжний рядки}
      i, j k:integer;
begin
 write(‘Введіть рядок : ’);readln(s);
 s:=s+’ ’;
 j:=0; ss:=’ ’;
 for i:=1 to length(s)-1 do
  if (s[i]<>’ ’) and(s[i+1]=’ ’)    {виділення слів}
  then begin
ss:=ss+s[i];
j:=j+1;a[1, j]:=ss;a[2, j]:=’0’;{у перший рядок}
ss:=’ ’;              { записуємо слово}
end; {0’ в другому рядку означає, що слово зустрілося вперше}
```

```
       else if s[i]<>' ' then ss:=ss+s[i];
      for i:=1 to j-1 do
        for k:=i+1 to j do
           if  (a[2,i]<>'1')  and(a[2,k]<>'1')  and(a[1,i]=a[1,k])
then a[2,k]='1';  {знайшли слова, що співпали}
      s:=' ';
      for i:=1 to j do
        if a[2,i]<>'1' then s:=s+a[1,i]+' ';
      writeln(`Результат : ',s);
     end.
```

Завдання 2. Залишити в рядку тільки перше входження кожного символу, взаємний порядок залишених символів зберегти.

```
program UPR8-2;
var s: set of char;
inp, res: string;
i: byte;
begin
   s:=[];
   res:= '';
   for i:= 1 to length(inp) do
         if not(inp[i] in s)
           then begin res:= res+inp[i];
                      s:= s+[inp[i]];
                end; end.
```

Завдання 3. Залишити в рядку тільки останнє входження кожного символу, взаємний порядок залишених символів зберегти.

```
program UPR8-3;
var inp, res: string;
i: byte;

begin
   res:= '';
   for i:= 1 to length(inp) do
      begin
         k:= pos(inp[i],res);
         if k<>0
            then delete(res,k,1);
         res:= res+inp[i];
      end;
end.
```

Завдання 4. Видати перші 100 000 натуральних чисел у випадковому порядку без повторень.

```
program UPR8-4;
var bset: array[0..12499] of byte;   { множина, бітовий
```

```
масив}
        ed: array[1..8] of byte;
        el,k: longint;
        kmp,bin: integer;
begin
   ed[1]:= 1;   генерація масиву
бітових одиниць}
   for k:= 2 to 8 do ed[k]:= ed[k-1] shl 1;
{-----------------------------------------------------}
   k:=0;
   randomize;   процедура активізації генератора випадкових
чисел}
   while k<100000 do
begin
el:= 1+random(99999);     випадкове    число    з    діапазону
0..99999}
        kmp:= el div 8;
        bit:= el mod 8;
        if bit=0 then bit:= 8;
           if bset[kmp]and ed[bit]=0  перевірка повторів}
        then begin inc(k);
                          writeln(el);
                          bset[kmp]:= bset[kmp]or ed[bit]
             end;
end
end.
End.
```

Завдання 5. Виравняти графу з прізвищами студентів по лівому краю.

```
Program UPR8-5;
Const
pr = '               '; {15 пропусків}
Var
fam fam1: string;
d: integer;
Begin
     writeln('Введіть прізвище студента');
     readln(fam);
     d := 15 - Length(fam);
     fam1 := fam + сміттю(pr,1,d);
     writeln('|' fam1, '|');
End.
```

На екран буде виведено:

Андрєєва С.В.

Змінна d в програмі визначає кількість пропусків, які треба приєднати до строкової змінної ***fam***, щоб одержати довжину рядка ***fam1***, рівну 15 символам.

Завдання для самостійної роботи

1. Задана пропозиція, що складається із слів, розділених одним або декількома пропусками. Упорядкувати слова пропозиції в алфавітному порядку.
2. Задана пропозиція, що складається із слів, розділених одним або декількома пропусками. Знайти щонайдовше слово в пропозиції.
3. Задана пропозиція, що складається із слів, розділених одним або декількома пропусками. Підрахувати кількість голосних букв в пропозиції.
4. Задана пропозиція, що складається із слів, розділених одним або декількома пропусками. Вивести на екран всі слова, перетворивши кожне при цьому таким чином: першу букву слова замінити на останню.
5. Задана пропозиція, що складається із слів, розділених одним або декількома пропусками. З'ясувати яка буква зустрічається найчастіше.
6. Напишіть програму підрахунку сумарного числа букв **‘a’** і букв **‘b’** в даній строковій змінній. Вивести на екран, яких букв більше.
7. Задано пропозицію **y**, що складається із слів-рядків. Перевірити, чи зустрічається дане слово **x** в пропозиції **y**.
8. Пропозиція містить букви латинського і російського алфавітів. Написати програму, яка виводить букви тільки латинського алфавіту у порядку їх проходження в пропозиції.
9. Дана пропозиція-рядок. Підрахувати кількість слів, що починаються з букви **‘a’**.
10. Написати програму, що підраховує, скільки разів в даному слові **x** зустрічається (як його частина) слово **y**.
11. Скласти програму, що виводить на екран перелік ліків (найменування виробу, вага, вартість).
12. Скласти програму, що виводить на екран студентську відомість (П.І.П., оцінки за три іспити, середній бал).
13. Скласти програму, що виводить на екран розклад руху занять судентів (дисципліни, час початку, зас закінчення, фамілія викладача).

14. Скласти програму, що виводить на екран анкетні дані учнів (П.І.П., рік народження, адреса, відомості про батьків).

15. Скласти програму, що виводить на екран список книг домашньої бібліотеки (автор, назва книги, видавництво, рік видання, вартість).

16. Скласти програму, що виводить на екран розклад іспитів і заліків (предмет, вид звітності, число, викладач).

17. Скласти програму, що виводить на екран відомості про студентів (П.І.П., курс, група, номер залікової книжки, середній бал).

18. Скласти програму, що виводить на екран зведення про періодичні видання (найменування видання, тираж, річна вартість).

19. Скласти програму, що виводить на екран графік відпусток (П.І.П., дата почала відпустки, дата виходу на роботу, кількість днів).

20. Даний текст-рядок з латинських букв і інших знаків. Підрахувати скільки букв, скільки знаків.

3.9 Контрольні питання до розділу 3

1. Назвіть структуровані типи даних в Паскалі.
2. Що таке масив?
3. Що визначає індекс одномірного масиву?
4. У програмі описаний масив ***VAR A: ARRAY (1..5,1..3) of real;***
5. Скільки рядків і стовпців містить матриця А?
6. Чому в математиці відповідає двомірний масив?
7. Чому в математиці відповідає одномірний масив?
8. Робота із строковими даними схожа на роботу з одномірним масивом даних. Яким повинен бути тип у цих даних?
9. Що є типом даних – рядок?
10. Дайте визначення поняттю «Множина».
11. Дайте визначення поняттю «Запису».
12. Дайте визначення поняттю «Файл».
13. Дано опис: ***VAR S: SET of char***. Опису якого типу структурованих даних воно відповідає?
14. Дані дві константи множинного типа А [1..6] і В[5,7,9,11]. Чому рівний результат їх перетину?
15. Дані дві константи множинного типа А [2..5] і В[1,8,3,5]. Чому рівний результат їх різниці ?
16. Дайте визначення поняттю «Процедура». Як вона описується у Pascal?
17. Дайте визначення поняттю «Функція». Як вона описується у Pascal?
18. У яких конструкціях існують локальні змінні, константи і типи?
19. Дайте визначення поняттю «Модуль».
20. Що описує в програмі текст USES CRT?
21. Дано опис : ***function fun(x:real):real.*** Опису яких даних воно відповідає?

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Немнюгин С.А. Учебник по Турбо Паскаль. С-Пб. “Питер”, 2003. – 560с.
2. Немнюгин С.А. Практикум по Турбо Паскаль. С-Пб. “Питер”, 2001. – 256с.
3. Турбо Паскаль 7.0 Самоучитель СПб. Питер. – 2002. – 416с.
4. Булах, І. Є. Комп'ютерне моделювання у фармації : навч. посіб. / І. Є. Булах, Л. П. Войтенко, І. П. Кривенко. - 2-ге вид., випр. - Київ : Медицина, 2017. - 208с.
5. Форкун Ю. В. Інформатика : навч. посіб. / Ю. В. Форкун, Н. А. Длугунович. - – Львів : Видавництво «Новий світ – 2000», 2018. – 464с.
6. Комп’ютери та комп’ютерні технології: навч. посіб. / Ю. Б. Бродський, К. В. Молодецька, О. Б. Борисюк, І. Ю. Гринчук. – Житомир : Вид-во «Житомирський національний агроекологічний університет», 2016. – 186с.
7. Герасевич В.А. Компьютер для врача. Самоучитель. – 2-е изд., перераб. и доп. – СПб.: БХВПетербург, 2004. – 512 с. 2. Г.Харт-Дэвис. Microsoft Windows XP Professional. Полное руководство./ Пер. с англ. – М.: СП ЭКОМ, 2004. – 816с.
8. Гельман В.Я. Медицинская информатика: практикум. – СПБ: Питер, 2001. – 480 с.
9. Ищейкина Ю.А., Лобач Н.В. Основы медицинской информатики: Учебное пособие. УМСА: Полтава, 2012. – 255с.
10. Силкова Е.В., Лобач Н.В. Медицинская информатика [Текст] : учебное пособие / Силкова Е.В., Лобач Н.В.; ВГУЗУ «УМСА ». – Полтава : ООО «АС МИ», 2015. – 233 с.

ДОДАТОК 1. Повідомлення компілятора про помилки

1 - вихід за межі пам'яті

2 - не вказаний ідентифікатор

3 - невідомий ідентифікатор

4 - повторний ідентифікатор

5 - синтаксична помилка

6 - помилка в дійсній константі

7 - помилка в цілій константі

8 - строкова константа перевищує розміри рядка

10 - неправильний кінець файлу

11 - рядок дуже довгий

12 - потрібен ідентифікатор типу

14 - невірне ім'я файлу

15 - файл не знайдений

16 - диск заповнений

20 - потрібен ідентифікатор змінної

21 - помилка у визначенні типу

25 - невірна довжина рядка

26 - невідповідність типів

27 - неправильний базовий тип відрізка типу

28 - нижня межа більше за верхню

29 - потрібен порядковий тип

30 - потрібна ціла константа

31 - потрібна константа

32 - потрібна ціла або дійсна константа

33 - потрібен ідентифікатор типу

34 - неправильний тип результату функції

36 - потрібен BEGIN

37 - потрібен END

38 - потрібен вираз типу Integer

40 - потрібен вираз типу Boolean

41 - типи операндів не відповідають оператору

42 - помилка у виразі

43 - невірне привласнення

44 - потрібен ідентифікатор поля

50 - потрібен оператор DO

54 - потрібен OF

57 - потрібен THEN

58 - потрібен DO або DOWNTO

59 - що невизначене випереджає опис

61 - невірне перетворення типу

62 - ділення на нуль

63 - невірний файловий тип

64 - немає можливості вважати або записати змінні даного типа

66 - потрібна строкова змінна

67 - потрібен вираз строкового типа

74 - типи констант і тип виразу оператора CASE не відповідають один одному

75 - потрібна змінна типу запис

76 - константа порушує межі

77 - потрібна файлова змінна

79 - потрібен вираз типу real або integer

84 - потрібне UNIT

85 - потрібно вказати ";"

86 - потрібно вказати ":"

87 - потрібно вказати ","

88 - потрібно вказати "("

89 - потрібно вказати ")"

90 - потрібно вказати "="

91 - потрібно вказати ":-"

92 - потрібне "[" або "(."

93 - потрібне "]" або ".)"

94 - потрібне "."

95 - потрібне ".."

96 - дуже багато змінних

97 - неправильна змінна циклу оператора FOR

98 - потрібна змінна цілого типа

100 - невідповідність довжини строкової змінної або константи

101 - невірний порядок полів

102 - потрібна константа строкового типа

103 - потрібна змінна типу integer або real

106 - попередній вираз повинен мати символьний тип

108 - недостатньо пам'яті для виконання програми

109 - немає можливості знайти файл .EXE

110 - модуль виконувати не можна

112 - константа оператора CASE знаходиться поза межами

114 - немає можливості викликати процедуру переривання

123 - дуже багато символи (більше 64 Кбайт)

124 - дуже великий розділ операторів (більше 24 Кбайт)

126 - файли повинні мати параметри VAR

127 - дуже багато умовні символи

130 - помилка в початкових умовних визначеннях

131 - заголовок не відповідає попередньому визначенню

132 - крити1 1ская помилки диска

133 - не можна обчислити даний вираз

134 - некоректне завершення виразу

135 - невірний специфікатор формату

136 - неприпустиме непряме посилання

137 - тут не допускається використання структурної змінної

138 - не можна обчислити без блоку System

139 - доступ до даного символу відсутній

140 - неприпустима операція з плаваючою комою

142 - повинні використовуватися змінна-процедура або функція

143 - неприпустиме посилання на процедуру або функцію

ДОДАТОК 2. Деякі повідомлення про помилки виконання програм

1 - не знайдений файл

3 - не знайдений маршрут

4 - дуже багато відкриті файли

5 - відмовлено в доступі до файлу

15 - неприпустимий номер дисковода

16 - не можна видалити поточний каталог

17 - не можна при перейменуванні указувати різні дисководи

100 - помилка читання диска

101 - помилка запису на диск

102 - файлу не привласнено ім'я

103 - файл не відкритий

104 - файл не відкритий для введення

105 - файл не відкритий для висновку

106 - невірний числовий формат

150 - диск захищений від запису

151 - невідомий модуль

152 - дисковод знаходиться в стані "не готовий"

153 - непізнана команда

154 - помилка в початкових даних

155 - при запиті до диска невірна довжина структури

156 - помилка при операції установки головок на диску

157 - невідомий тип носія

153 - сектор не знайдений

159 - кінчився папір на пристрої друку

160 - помилка при записі на пристрій

161 - помилка при читанні з пристрою

162 - збій апаратури

200 - ділення на нуль

201 - помилка при перевірці меж

202 - переповнювання стека

203 - переповнювання динамічної області пам'яті

204 - недійсна операція посилання

205 - переповнювання операції з плаваючою комою

206 - зникнення порядку при операції з плаваючою комою

207 - неприпустима операція з плаваючою комою

ДОДАТОК 3. Табліця кодування