МІНІСТЕРСТВО АГРАРНОЇ ПОЛІТИКИ І ПРОДОВОЛЬСТВА УКРАЇНИ
Аграрний коледж управління і права
Полтавської державної аграрної академії

МОВА ПРОГРАМУВАННЯ PASCAL

ЕЛЕКТРОННИЙ ПОСІБНИК

для самостійної роботи студентів аграрних ВНЗ І-ІІ рівнів акредитації

напрям підготовки 0305 Економіка та підприємництво
спеціальність 5.03050201 Інформаційна діяльність підприємства

Укладач: к.пед.н.Кононец Наталія Василівна, http://informatika-resurs.jimdo.com/

Полтава
2014

Зміст

Основні поняття та означення 3
Компілятор та інтерпретатор 7
Мова програмування Pascal 8
Поняття алгоритму 10
Елементи мови Pascal 20
Типи даних 23
Арифметичні операції, функції, вирази. Арифметичний оператор присвоювання 26
Оператори ВВОДУ-ВИВОДУ 27
Форматування даних 29
Керування порядком обчислень. Розгалуджені алгоритми 30
    Оператори повторів 33
    Циклічні алгоритми 36
Процедурно-орієнтоване програмування. Функції користувача 38
    Процедури та функції 40
    Використання підпрограм 43
    Рекурсія 44
Масиви. Одновимірні масиви 46
    Багатовимірні масиви 50
    Рядки 54
Навчальні ресурси 56

Основні поняття та означення

ВИЗНАЧЕННЯ МОВ ПРОГРАМУВАННЯ

Мова – це сукупність засобів і правил для фіксації повідомлень і їх передавання. Мови, призначені для фіксації алгоритмів, називають алгоритмічними мовами.Алгоритмічні мови, призначені для фіксації алгоритмів, розрахованих на виконання комп'ютером, називають мовами програмування.

Мова має свій *синтаксис* і *семантику*.

Синтаксис - сукупність правил, що описують правильні конструкції мови.

Семантика - опис правильного виконання конструкцій мови.

Загальна схема підготовки програм до виконання

Підготовка задачі до її рішення за допомогою обчислювальної техніки містить наступні дії:

1) Отримати завдання з визначенням мети і умов рішення задачі..
2) Уважно прочитати, визначити мету, вхідні і вихідні дані.
3) Вибрати метод рішення.
4) Скласти алгоритм рішення.
5) Розробити програму на обраній мові програмування згідно алгоритму.
6) Реалізувати програму на комп'ютері:

- Ввести текст програми за допомогою текстового редактора і запам'ятати його в файлі. Отриманий файл буде називатись **висхідним файлом програми** і мати текстовий формат. Ім'я файлу призначає програміст, а розширення файлу призначається системою програмування. Для мови Pascal висхідний файл має розширення **.pas.**
- Виконати трансляцію програми. Результатом буде двійковий файл який матиме теж ім'я, що і висхідний файл, а розширення **.obj.** Отриманий файл називається об'єктним.файлом.
- Щоб програма могла виконуватись її необхідно адаптувати до середовища. Для цього виконується редагування зв'язків програми (побудова, компонування). У результаті компонування формується двійковий файл, що має назву завантажувальний файл, ім'я таке як у висхідного файлу і розширення **.exe.**
- Запустити програму на виконання.

Для полегшення роботи програмістів для підготовки програм використовуються інтегровані середовища (системи програмування) до складу яких входять наступні елементи:

- Мова програмування,
- Текстовий редактор,
- Транслятор,
- Компоновщик,
- Програма лагодження і інші.

ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ TURBO PASCAL

Система меню інтегрованого середовища

Основні пункти меню:

- **File** (файл) – дії з файлами, вихід з системи

- **Edit** (редагування) – відновлення, операції з буфером обміну;
- **Search** (пошук) – пошук тексту, процедури, функції, помилки;
- **Run** (запуск) – запуск програми на виконання;
- **Compile** (компіляція) – компіляція програми;
- **Debug** (лагодження) – лагодження програми;
- **Tools** (інструменти) – запуск допоміжних програм;
- **Options** (опції) – установка параметрів середовища;
- **Window** (вікно) – робота з вікнами;
- **Help** (допомога) – звертання до довідкової служби.

Меню опції *File*
NEW - створює і відкриває нове вікно редактора з іменем NONAMExx.pas. Номер xx залежить від кількості відкритих вікон без ім'я.
OPEN - відкриває діалогове вікно для відкриття файлу, якщо фай вибрано - створює нове вікно редактора і розміщує в ньому текст вибраного файлу.
SAVE - записує уміст поточного вікна редактора в файл. Якщо це вікно з ім'ям NONAMExx.pas, відкривається діалогове вікно для вибору місця збереження тексту програми, якщо файл було відкрито чи вже збережено, текст зберігається за тою ж адресою.
SAVE AS - Використовується для збереження тексту програми в іншому місці або з іншим ім'ям.
SAVE ALL - Записує уміст всіх вікон у відповідні файли.
CHANGE DIR - Дозволяє змінити поточний каталог користувача.
PRINT - друк тексту програми або вивід його в текстовий файл
PRINTER SETUP - налагоджує середовище для друку файлу
DOS SHELL - Забезпечує тимчасовий вихід в середовище DOS.
EXIT - Завершення роботи

Меню опції *Edit*
UNDO - відновлення останніх змін.
REDO - відміна дії команди UNDO.
CUT - вилучає виділений блок тексту з вікна редактора і заносить в буфер обміну;
COPY - копіює виділений блок тексту з вікна редактора і заносить в буфер обміну;
PASTE - копіює уміст буферу обміну в вікно редактора, починаючи з позиції курсору;
CLEAR - вилучає виділений блок тексту з вікна редактора, але не заносить в буфер обміну;
SHOW CLIPBOARD - Показує уміст буферу обміну.

Меню опції *Search*
FIND - забезпечує пошук потрібного фрагменту тексту в вікні редактора і переміщує на нього курсор;
REPLACE - забезпечує пошук потрібного фрагменту тексту в вікні редактора і Замінює його на новий вказаний текст;
SEARCH AGAIN - повторює пошук і заміну фрагменту для раніше встановлених параметрів;
GO TO LINE NUMBER - переміщує курсор на рядок з вказаним номером;
SHOW LAST COMPILE ERROR – показує рядок програми, в якому була помилка при останній компіляції;

FIND ERROR - відшукує в тексті програми рядок, в якому була помилка при виконанні програми;
FIND PROCEDURE - в режимі лагодження відшукує потрібну процедуру або функцію.

Меню опції *Run*
RUN - виконує компіляцію, компонування і запуск програми на виконання з вікна редактора;
GO TO CURSOR - починає чи продовжує виконання програми в режимі лагодження до позиції курсору;
TRACE INTO - починає чи продовжує режим лагодження . Виконання зупиняється перед кожним оператором. При виклику підпрограми процес також зупиняється в підпрограмі. Для продовження натиснути пункт меню або клавішу F7
STEP OVER - Виконується так як попередня команда, але перехід до підпрограми не відслідковується. Продовження – клавіша F8.
PROGRAM RESET - відміняються всі установки лагодження ;
PARAMETERS - дозволяє задати текстовий рядок параметрів, що передаються програмі.

Меню опції *Compile*
COMPILE - компілює програму або модуль ;
MAKE -створює програму, яка може включати файли або звертання до нестандартних бібліотек. Спочатку компілюється файл вказаний в опції **PRIMARY FILE**, його не вказано – то з вікна редактора. Якщо в процесі компіляції зустрічається об'ява нестандартного модуля і його немає відкомпільованого, або в його тексті були зроблені зміни, то він також компілюється. Якщо текстового файла модуля не знайдено, то його компільований варіант не змінюється.
BUILD - як і в попередньому разі, але нестандартні модулі компілюються завжди.
DESTINATION - вказує де розміщувати вихідний файл: в пам'яті чи на диску.
PRIMARY FILE - Задає ім'я головного файлу.
CLEAR PRIMARY FILE - знищує ім'я головного файлу;
INFORMATION - показує статистику програми;

Меню опції *Debug*
BREAKPOINTS - дозволяє передивитись всі контрольні точки та вилучити всі чи потрібну;
CALL STACK - показує уміст стеку;
REGISTER - показує значення всіх регістрів процесора;
WATCH - робить активним вікно лагодження;
OUTPUT - робить активним вікно програми ;
USER SCREEN - робить активним вікно програми і розкриває його на весь екран;
EVALUATE/MODIFY - дозволяє передивитись значення змінних та змінити їх;
ADD WATCH - дозволяє задати в вікні перегляду потрібні змінні;
ADD BREAKPOINT - дозволяє додати контрольну точку зупинки;

Меню опції *Options*
DIRECTORIES - встановлюються функціональні каталоги:
EXE & TPU - каталог для збереження результуючих файлів;
Include directories - каталоги, в яких містяться файли, що підключаються;
Unit directories - каталоги, де відшукуються TPU-файли.

Меню опції *Window*
TITLE - розміщення вікон так, щоб вони мали однакові розміри;
CASCADE - розміщення вікон так, щоб було видно рамки вікон;
CLOSE ALL - закрити всі вікна;
REFRESH DISPLAY - вилучення результатів програми з вікна програми;
SIZE/MOVE - переміщення вікна по екрану та зміна його розмірів;
ZOOM - розкрити вікно на весь екран;
NEXT - перейти до наступного вікна;
PREVIOS - перейти до попереднього вікна;
CLOSE - закрити активне вікно;
LIST - виводить на екран перелік всіх вікон. Можна вибрати потрібне і перейти в нього.

Компілятор та інтерпретатор

Сучасні мови програмування подібні до загально людської мови і описують алгоритм за допомогою визначених операторів. Така програма далі потребує перекладу на мову машинних кодів, тому для алгоритмічної мови обов'язково треба використання спеціальної програми–перекладача – **транслятора**. Розрізняють такі види трансляторів:

- **транслятор-інтерпретатор** моделює обчислювальну машину, для якої машинною мовою є дана алгоритмічна мова. Інтерпретатор транслює кожний оператор в деякий проміжний код, інтерпретує його через одну або декілька машинних команд та зразу виконує ці команди. При інтерпретації програма в машинних кодах не зберігається, тому при кожному запуску вихідної програми на виконання її треба по крокам транслювати знову. Головна риса інтерпретатора є простота.

- **транслятор-компілятор** – програма, яка сприймає вихідну програму на алгоритмічній мові та перетворює її в програму на мові в машинних командах. Компілятор обробляє програму в цілому, і отриманий об'єктний файл завантажується на виконання. За допомогою компілятора можна створити файл з розширенням .exe, який виконується самостійно, без відповідного середовища програмування.

Компіляцію можна порівняти з письмовим перекладом, а інтерпретацію – з синхронним перекладом мови. За способом реалізації трансляторів алгоритмічна мови поділяються таким чином:

- Pascal, C, Fortran – транслятор-компілятор,
- Basic – транслятор-інтерпретатор.

МОВА PASCAL

Автором мови є Ніколас Вірт - професор, директор Інституту Інформатики Швейцарської вищої політехнічної школи. Перше повідомлення про мову було опубліковано в 1971 році. Мова була розроблена для навчання студентів програмуванню. Але з'ясувалось, що мова має багато достоїнств і стала використовуватись професіональними програмістами в професійній діяльності. З часом мова удосконалювалась, виникали різні „діалекти" мови. На сьогоднішній день мова також використовується в професійній діяльності.

ЕЛЕМЕНТИ МОВИ

Алфавіт

Алфавіт - це сукупність знаків, що використовуються для побудови конструкцій мови.
Алфавіт мови Pascal складають:
- латинські букви великі і маленькі (A..Z, a..z);
- цифри (0..9);
- спеціальні символи (= # + - . , : ; ‘ @ $ * ( ) [ ] { } / < > пропуск);
- службові слова (begin end for until . . .).

Великі і малі літери не розрізняються.

Числа

Для зображення чисел по умовчанню використовується десяткове подання.
Числа можуть бути цілими і дійсними.
Цілі числа можуть мати знак. Знак *плюс* може опускатись. Наприклад: 234, -34.
Можна використовувати цілі числа в 16-річному форматі. У цьому разі значення повинно починатись із символу $. Наприклад: $567, $6F41, -$216.
Дійсні числа можуть бути представлені в звичайній формі (23.4, -34.8123, 0.0034) або в експоненціальній формі:
Математичне представлення числа в експоненціальній формі:

$$7230 = 7.23\times10^{3}\ ,\ 0.00067 = 6.7\times10^{-4}.$$

Відповідний запис на мові Pascal: 7.23E3 6.7E-4
Не можна дійсне число починати з крапки (.789), потрібно записувати 0.789.

Рядки

Рядок - це послідовність довільних символів, обмежених апострофами (одинарними лапками). Наприклад:
‘Розум поперед справи’ , ‘Золоті руки у того, хто навчався добре’

Константи

Константи – це величини, які не змінюються.
Константи можуть бути:

- числовими (45, -62.7896 $57),
- рядковими (‘Очі - дзеркало душі.’),
- символьними (‘a’, ‘6’, ‘%’),
- логічними (true, false)

Операції

Операції - це дії над заданими величинами. Величини, що задіяні в операції, називаються операндами. Наприклад, операція додавання чисел 2 і 3 записується 3 + 2. Числа 2 і 3 будуть операндами.

Операції можуть бути трьох типів: арифметичні, логічні і операції відношень.

Арифметичні операції:

+ - додавання,
- - віднімання,
* - множення,
/ - дійсне ділення,
div - ціле ділення
mod - взяття залишку

У операціях + - * / операндами можуть бути і цілі і дійсні числа. Результат операції / завжди дійсне число. Операндами операцій div і mod можуть бути тільки цілі числа . Результат цих операцій також цілі числа.
Наприклад: 17 div 4 = 4, 20 div 4=5, 17 mod 4 = 1, 20 mod 4 = 0.

Якщо виконувати ділення в стовпчик, то частне буде результатом операції div, а залишок - результатом операції mod.

Операції піднесення до степеня у мові Pascal нема. Для цього можна використовувати тотожність:

$x^y = e^{y\ln(x)}$ або за правилами мови = exp(y*ln(x))

Для рядків можна виконувати операцію додавання. Результатом буде рядок, що створюється шляхом дописування другого рядка в кінець першого. Наприклад:

‘Хліб ’ + ‘всьому голова.’ = ‘Хліб всьому голова.’

Поняття алгоритму

Алгоритм - це опис послідовності дій, які приводять до досягнення визначеної мети.

Вірно розроблений алгоритм повинен мати наступні основні **властивості**:

- **Дискретність**. Алгоритм розв'язання задачі повинен складатися з послідовності окремих кроків — відокремлених одна від одної команд (указівок), кожна з яких виконується за кінцевий час. Тільки закінчивши виконання однієї команди, виконавець переходить до виконання іншої.
- **Визначеність** (однозначність). Кожна команда алгоритму однозначно визначає дії виконавця і не припускає подвійного тлумачення. Суворо визначеним є й порядок виконання команд.
- **Формальність**. Будь-який виконавець, який володіє заданою системою команд, може виконати заданий алгоритм, не вникаючи в суть задачі.
- **Результативність**. Виконання алгоритму не може закінчуватися невизначеною ситуацією або зовсім не закінчуватися. Будь-який алгоритм передбачає, що його виконання при допустимих початкових даних за кінцеве число кроків приведе до очікуваного результату.
- **Масовість**. Алгоритм має передбачати можливість зміни початкових (вхідних) даних у деяких допустимих межах і можливість використання його для розв'язання задач одного класу (універсальність алгоритму).

Саме через ці властивості часто дається визначення **поняття алгоритму як скінченної однозначно визначеної послідовності операцій, формальне виконання якої приводить до розв'язання певної задачі за кінцеве число кроків.**

Виконавець алгоритму

Виконавцем алгоритму може бути людина, тварина, ЕОМ, система людина-машина, верстат-автомат, робот тощо, тобто ті хто розуміє та може виконати команди алгоритму.
Точне визначення, що таке виконавець, дати дуже складно, та в цьому й немає потреби. Важливо зрозуміти основні характеристики виконавця: середовище, елементарні дії, систему команд, відмови.
Середовище - це місце перебування виконавця. Кожен виконавець може виконувати команди лише з деякого строго заданого списку - *системи команд виконавця*. Для кожної команди повинні бути задані умови застосування, тобто в яких станах середовища може бути виконана команда, й описані результати її виконання.
Виконавця можна представити у вигляді деякого пристрою, керування яким спричиняє виконання тієї чи іншої команди. Після виклику команди виконавець здійснює відповідну елементарну дію. Важливо зазначити, що в цьому процесі нас більше цікавить результат, а не механізм виконання команди. Відмови виконавця виникають за умови виклику команди в неприпустимому для даної команди стані середовища.
Ситуації, при яких виникає відмова, різні для різних команд виконавця (наприклад, ділення на нуль, відсутність принтера при виведенні інформації на друк). При виконанні деяких команд відмови ніколи не виникають (наприклад, обчислення виразу 2 + 2).
У деяких випадках виконавцем алгоритмів є людина. Наприклад, якщо йдеться про правила, інструкції, функціональні та посадові обов'язки тощо. Але людина далеко не єдиний

виконавець алгоритмів. Роботи-маніпулятори, верстати з програмним управлінням, жива клітина і навіть тварини в цирку виконують різноманітні алгоритми, в тому числі й ті алгоритми, які людина не може виконати.
Що ж таке виконавець? Спрощено виконавця можна уявити собі як деякий пристрій керування, з'єднаний з набором інструментів. Пристрій керування розуміє алгоритми і організовує їх виконання, користуючись при цьому відповідними інструментами. А інструменти, сприймаючи команди керуючого пристрою, виконують дії. Якщо людину розглядати як виконавця, то можна провести таку аналогію: *мозок - пристрій керування, руки, ноги, очі, ніс тощо - інструменти*. У роботів-маніпуляторів, верстатів з програмним управлінням, комп'ютерів пристроєм керування є процесор, а набір інструментів залежить від того, для розв'язування яких задач призначений той чи інший виконавець.

Формальне виконання алгоритму

Під формальним виконанням алгоритму розуміється таке його виконання, коли сам виконавець не знає ні постановки задачі, ні змісту отриманих результатів, але, чітко виконуючи усі дії, записані в алгоритмі, досягає результату.
Найчастіше алгоритми виконуються саме формальним чином. Формальними виконавцями є користувачі різноманітних побутових електричних приладів, учасники спортивних ігор, які дотримуються правил цих ігор, учні, що застосовують у своїх творчих роботах мовні правила, виконують завдання з математики чи фізики, використовуючи відомі формули, закони та методи розв'язування різних типів задач. Продовжуючи цю думку, комп'ютер також слід вважати формальним виконавцем алгоритмів при виконанні ним програм.

Способи представлення алгоритму

1. Словесні.
2. Словесно-формульні.
3. Графічні.
4. Скінчений набір кодів.

При складанні алгоритмів можна поєднувати різні форми подання алгоритмів.

Процес алгоритмізації — це визначення елементарних дій та порядку їх виконання для розв’язання поставленого завдання. Існують різні способи запису алгоритмів (словесний, формульно-словесний, метод блок-схем, програмний та ін.), які застосовуються для представлення алгоритму у вигляді, що однозначно розуміється і розробником, і виконавцем алгоритму.
Для опису алгоритмів людина часто користується природною мовою, але для запису багатьох алгоритмів природна мова виявилась незручною, тому виникла необхідність у створенні штучних мов, наприклад мови математичних формул, хімічних процесів тощо. Існує спеціальна навчальна алгоритмічна мова, яка була створена для запису алгоритмів на папері; вона використовує слова природної мови, але має більш жорстку структуру. Найбільше поширення для запису логічної структури алгоритмів отримали графічні (структурні) схеми,

які спрощують складання та аналіз алгоритму, полегшують перехід від запису алгоритму до написання програми.

Графічна схема (блок-схема) алгоритму — це графічне зображення алгоритму у вигляді спеціальних блоків з необхідними словесними поясненнями. Кожний етап алгоритму представляється у вигляді геометричної фігури (блоку), що має певну форму в залежності від характеру операції. Блоки на схемі з'єднуються стрілками (лініями зв'язку), які визначають послідовність виконання операцій та утворюють логічну структуру алгоритму.
Важливою особливістю базових структур алгоритмів є те, що вони мають один вхід і один вихід, що дозволяє при відносній незалежності конструювати окремі блоки алгоритмів, а потім окремо розроблені структури з'єднувати між собою (вихід однієї базової структури сполучається із входом іншої). Весь алгоритм представляє лінійну послідовність базових структур.

Аргументи, результати, проміжні величини
Для роботи багатьох алгоритмів треба задавати початкові значення. Ці значення передаються в алгоритм за допомогою аргументів.
Аргумент - це величина, значення якої необхідно задати для виконання алгоритму.
Проте є алгоритми, що не потребують жодних початкових значень для свого виконання. Пізніше ми ознайомимося з такими алгоритмами. Однак немає жодного алгоритму, що не дає ніякого результату. Справді, яка ж потреба в такому алгоритмі? Прикладом різноманітності результатів роботи програм є ігрові комп'ютерні програми. У цих програмах дані обробляються та певним чином перетворюються у графічні і звукові образи.

Результат - це величина, значення якої отримується внаслідок виконання алгоритму.
Під час складання багатьох алгоритмів виникає необхідність крім аргументів і результатів використовувати ще й додаткові величини. Введення в алгоритм таких величин задає автор алгоритму.

Проміжна величина - це величина, яка додатково вводиться в процесі розробки алгоритму.

Розглянемо алгоритми розв'язування задачі

Задача. Є посудина місткістю 8 літрів, яка заповнена рідиною, і дві порожні посудини місткістю 5 л і 3 л. Потрібно одержати в одній з посудин 1 літр рідини і повідомити в якій.

Розглянемо виконавця, який має таку систему команд:

1. Перелити рідину з однієї посудини в іншу.
2. Наповнити одну з посудин рідиною з іншої посудини.
3. Вивести повідомлення.

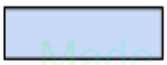
Для цього виконавця алгоритм розв'язування цієї задачі буде таким:
Вхідні дані: посудина місткістю 8 літрів, яка заповнена рідиною,
дві порожні посудини місткістю 5 л і 3 л.

1. Наповнити 3-літрову посудину з 8-літрової.
2. Перелити з 3-літрової посудини в 5-літрову.
3. Наповнити 3-літрову посудину з 8-літрової.
4. Наповнити 5-літрову посудину з 3-літрової.
5. Вивести повідомлення: «1 л одержано в 3-літровій посудині».

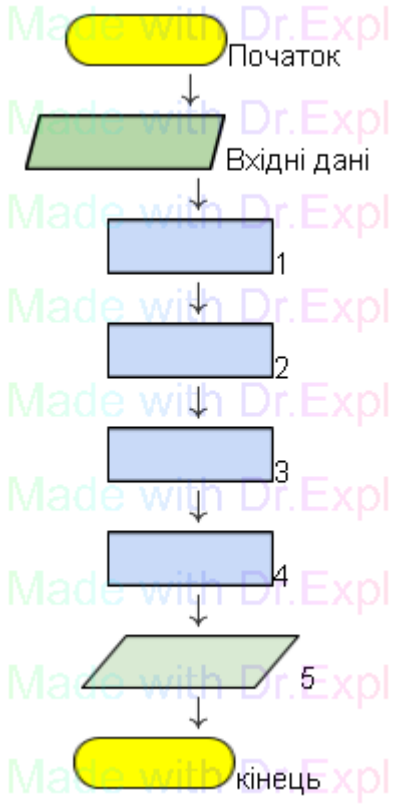
Наведений алгоритм записано у вигляді послідовності команд, кожна з яких має свій порядковий номер і записана українською мовою, тобто мовою людського спілкування. Така форма подання алгоритму називається ***словесною***.

У такому алгоритмі команди виконуються послідовно, тобто після виконання кожної команди виконується команда, яка записана наступною.

Для складніших алгоритмів словесна форма представлення алгоритму не завжди є зручною і наочною. Тому, крім такої форми подання, часто використовують ***графічну форму подання алгоритму***, або подання алгоритму у вигляді **блок-схеми.**
У блок-схемі алгоритму кожна команда записується в геометричній фігурі (блоці) певного вигляду. Блоки з'єднуються між собою стрілками, які вказують напрям переходу для виконання наступної команди алгоритму.

Назва	Зображення	Що позначає
Термінатор		Початок або кінець алгоритму
Процес		Виконання однієї або кількох команд
Дані		Введення або виведення даних (результатів)
Рішення		Прийняття рішення залежно від результату перевірки, умови, вказаної всередині цього елемента

Ось так буде виглядати **блок-схема** алгоритму розв’язання сформульованої вище задачі.



Задачі з теми: “Способи подання алгоритмів”

I варіант

1. Складіть алгоритм приготування бутерброда з сиром. Подайте його у словесній формі та у вигляді блок-схеми.
2. (Ця стародавня задача вперше трапляється в математичних рукописах VIII ст.) Човняру потрібно перевезти в човні через річку вовка, козу і капусту. У човні, крім човняра, вміщується або тільки вовк, або тільки коза, або тільки капуста. На березі не можна залишати козу з вовком або козу з капустою. Складіть алгоритм перевезення. Подайте його в словесній формі та у вигляді блок-схеми.
3.* Необхідно приготувати суп з концентрату. У нашому розпорядженні є пісочні годинники на 3 хв і 8 хв. Складіть алгоритму відліку часу для приготування супу, якщо його потрібно готувати рівно 7 хв. Подайте його в словесній формі та у вигляді блок-схеми.

II варіант

1. Напишіть алгоритм заварювання чаю. Подайте його у словесній формі та у вигляді блок-схеми.
2. Двом солдатам потрібно переправитися з одного берега річки на інший. Вони побачили двох хлопчиків на маленькому човні. У ньому можуть переправлятися або один солдат, або один чи двоє хлопчиків. Складіть алгоритм переправлення солдатів. (Після переправлення солдатів човен повинен залишитися у хлопчиків.) Подайте його в словесній формі та у вигляді блок-схеми.
3. Необхідно приготувати суп з концентрату. У нашому розпорядженні є пісочні годинники на 3 хв і 8 хв. Складіть алгоритму відліку часу для приготування супу, якщо його потрібно готувати рівно 10 хв. Подайте його в словесній формі та у вигляді блок-схеми.

ЛІНІЙНІ АЛГОРИТМИ

В лінійних алгоритмах всі операції виконуються послідовно в порядку їх розташування в алгоритмі.

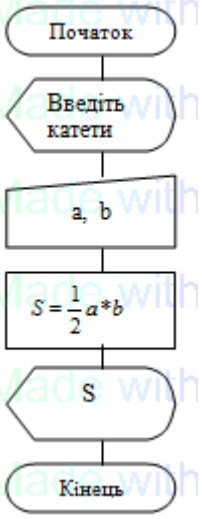
Такий порядок називається природнім. Наприклад: розробити алгоритм розрахунку площини прямокутного трикутника.

З математики відомо, що таку площину можна визначити як:

$S = 0.5 * a * b$, де a,b - катети трикутника.

Таким чином, спочатку необхідно ввести значення катетів, визначити площину і вивести результат. Це буде словесно-покроковий алгоритм.

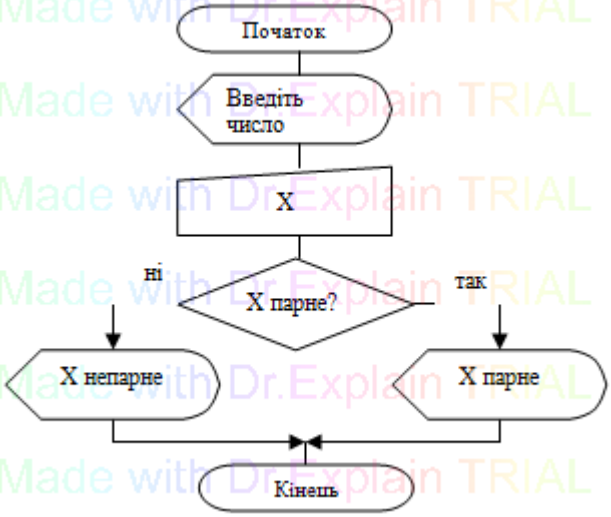
Схема алгоритму буде наступною:



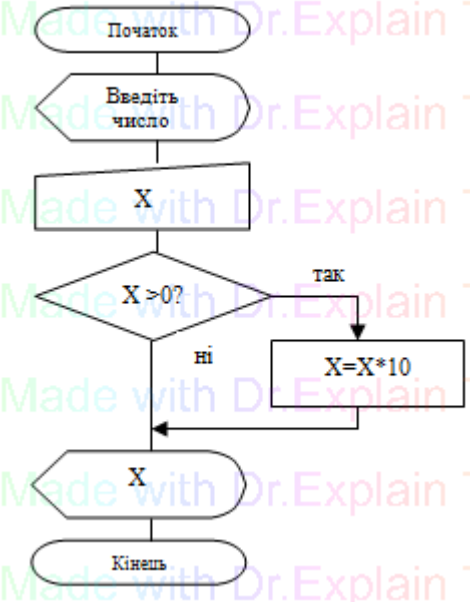
РОЗГАЛУДЖЕНИЙ АЛГОРИТМ

Алгоритм називається розгалуженим, якщо в результаті перевірки деякої умови може бути вибрано один з кількох можливих шляхів подальших дій.

Приклад 1 (виконання дій в двох випадках): визначити, чи є задане число парним.



Приклад 2 (виконання дій в одному випадку): якщо задане число є додатнім, то збільшити його в 10 разів.



ЦИКЛІЧНИЙ АЛГОРИТМ

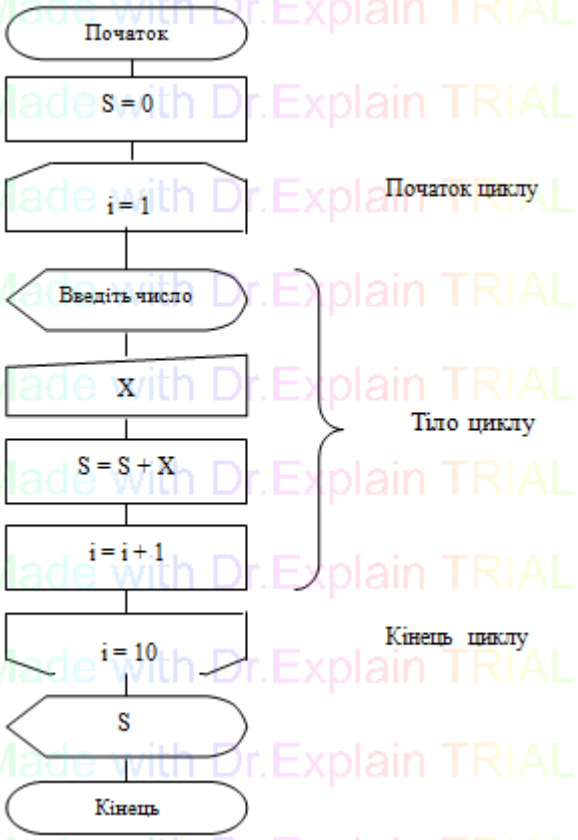
Циклічні алгоритми використовуються в тому разі, коли виникає необхідність багаторазового повтору однотипних дій при різних значення параметрів, що визначають ці дії.
Ділянки програми, що реалізовують повтори, називаються циклами. Кожен цикл має початок, де можуть формуватись висхідні дані, і кінець, де перевіряється умова завершення циклу. Дії, що повторюються в циклі, називаються тілом циклу.

Циклічні алгоритми можуть бути двох типів: з заданим числом повторів і з зарані не відомим числом повторів.

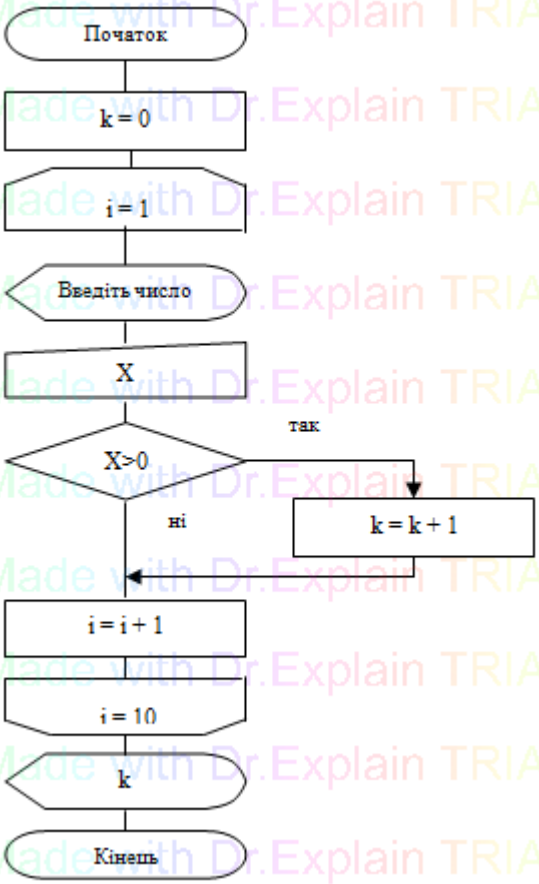
Приклад 1. Знайти суму 10 введених значень.

Міркуємо: накопичення суми необхідно починати з нуля, тілом циклу будуть дії по вводу наступного числа і додавання його до суми. Тіло циклу необхідно повторити 10 разів, ці

рази необхідно рахувати. Для їх підрахунку введемо змінну, наприклад „i”, яка повинна змінюватись від 1 до 10, збільшуючись на 1 після кожного циклу. Умовою завершення циклу буде значення i=10. Результатом буде накопичена сума, яку необхідно вивести.



Приклад 2. Знайти кількість позитивних значень в послідовності 10 введених значень.



Питання для самоконтролю

1. Чому алгоритм має властивість „визначеність”.
2. Чому алгоритм має властивість „масовість”.
3. Записати алгоритм переходу вулиці з двостороннім рухом словесно-покроково.
4. Записати той же алгоритм за допомогою схеми алгоритму.
5. Який алгоритм називають лінійним.
6. Який алгоритм називають розгалуженим.
7. Який алгоритм називають циклічним.
8. До якого типу алгоритму буде відноситись знаходження швидкості руху пішоходу, якщо відомо який він пройшов шлях і за який час.
9. До якого типу алгоритму буде відноситись визначення, чи є два заданих числа однієї парності.
10. До якого типу алгоритму буде відноситись побудова значень функції з заданим кроком на заданому проміжку.
11. Розробити алгоритм визначення площини прямокутного трикутника, якщо відомі його катети.
12. Розробити алгоритм визначення довжини кола, якщо відома його площина.
13. Розробити алгоритм визначення, чи задані числа кратні один одному.
14. Розробити алгоритм визначення, чи задані два числа впорядковані по зростанню.
15. Розробити алгоритм знаходження кількості парних чисел в послідовності 10 заданих чисел.

16. Розробити алгоритм знаходження кількості введених латинських букв і цифр в послідовності 15 заданих символів.
17. Розробити алгоритм знаходження найбільшого в лексикографічному значенні виразу в послідовності 10 заданих фраз.
18. Розробити алгоритм знаходження кількості введених позитивних і негативних значень окремо в послідовності 20 введених значень.
19. Розробити алгоритм знаходження кількості введених негативних значень і суми позитивних значень в послідовності 20 введених значень.
20. Розробити алгоритм знаходження максимального значення серед чисел, що входять в проміжок [5,15] в послідовності 20 введених значень.
21. Які мови називають мовами програмування.
22. Як називається програма, за допомогою якої виконується переклад програми з алгоритмічної мови на мову, зрозумілу комп'ютеру?
23. Що означає синтаксис мови.
24. Що означає семантика мови.
25. Які дії необхідно виконати, щоб підготувати видане завдання на розробку програми до її реалізації на комп'ютері.

Елементи мови Pascal

СТРУКТУРА ПРОГРАМИ

Програма, записана мовою Pascal, складається з заголовку і блоку.

Заголовок призначається для указівки імені програми.

Формат заголовку: program <ім'я програми>;

Наприклад, program MyFunk;

Блок складається з **описової** і **виконавчої** частин.

1. Описова частина

В описовій частині описується все те, що буде використано в виконавчий частині. Описова частина може складатись з кількох розділів. Їх послідовність не має значення. Не є помилкою, якщо розділи повторюються. Кожен розділ має своє ім'я і описує елементи програми. Кожен опис закінчується символом **„;".**

Розділ опису констант

Ім'я розділу const.

Використовується для опису іменованих констант. Константи, яким призначене символічне ім'я, називаються ***іменованими константами***.

Формат опису:

```
Const
<ім'я константи>=<значення>;
```

Наприклад:

```
Const
Kol=10;
Name='Наташа';
T=true;
```

В подальшому в тексті програми використовують не значення константи, а її ім'я. Використання іменованих констант робить програму більш наочною, забезпечує її простіше модифікування.

Розділ опису типів

Ім’я розділу type. Використовується для опису типів, визначених користувачем.

Формат опису:

```
Type
<ім’я типу> = <опис типу>|<перелік значень типу>;
```

Наприклад:

```
Type
Color=(red,white,blue,yellow);
Mebel=(ctol,ctul,shkaph);
```

<u>Розділ опису змінних</u>

Ім’я розділу var.
Використовується для опису всіх змінних, які будуть використовуватись в виконавчий частині..

Формат опису:

```
Var
<ім’я змінної> : <ім’я типу>|<опис типу>;
```

Наприклад:

```
Var
A,b:integer;
X,y:real;
T:Boolean;
C:byte;
DayWork:(Monday, Tuesday, Wednesday, Thursday, Friday);
```

<u>Розділ опису процедур і функцій</u>
В цьому розділі приводяться тіла процедур і функцій, які включені в програму. Процедури і функції розглядатимуться далі.

---

2. Виконавча частина

Виконавча частина являється тілом програми. Вона складається з одного розділу – розділу операторів.
Формат розділу:

Begin

<оператори>

end.

Коментарі

В будь-якій частині програми можна використовувати коментарі. **Коментар** - це текст, що пояснює призначення програми та її елементів і логіку програми.

Коментарі можна розміщувати в окремих рядках або в рядку після оператора. Коментар може розміщуватись в декількох рядках. Коментарі з обох боків обмежуються фігурними дужками { } або символами (* *).

Типи даних

Щоб вірно трактувати різні дані необхідно вказати його тип.

Тип даних визначає область допустимих значень для елементів цього типу і операції, які можна виконувати над елементами даного типу.

Типи даних поділяються на **прості** і **складні**.

- Прості типи не мають внутрішньої структури і служать для відображення простих фактів. До простих типів відносяться **числові, символьний, логічний, строковий, адресний, перелічений і обмежений типи**.

- Складні типи мають внутрішню структуру, яка складається з сукупності простих типів. До складних типів відносяться **масиви, множини, записи, файли, об'єкти, посилання і процедурний тип**.

Прості типи

Прості типи поділяються на **стандартні** і **визначені користувачем**.

1) Стандартні типи

Числові типи. Поділяються на цілі і дійсні типи.

Цілі типи:

Ім'я типу	Опис	Область значень	Розмір в b
Byte	коротке ціле без знака	0 . . 255	1
ShortInt	коротке ціле з знаком	-128 . . 127	1
Word	ціле без знака	0 . . 65535	2
Integer	ціле з знаком	-32768 . . 32767	2
LongInt	довге ціле з знаком	-2147483648 . . 2147483647	4

Дійсні типи:

Ім'я типу	Опис	Кількість значущих цифр	Діапазон порядку	Розмір в b
Real	Дійсне	11 .. 12	-39 .. 38	6
Double	Дійсне подвійної точності	15 .. 16	-324 .. 308	8
Single	Дійсне одинарної точності	7 .. 8		4
Extended	Дійсне високої точності	19 .. 20	-4951 .. 4932	10
Comp	Ціле в формі дійсного	19 .. 20	$-2^{63}+1$ .. $2^{63}-1$	8

Символьний тип

Ім'я типу Char. Значення займає 1 байт.

Це тип даних, що складається з одного символу. Запис значення цього типу є символ, взятий в апострофи: ‘a’, ‘8’.
Символ можна записати, використовуючи значення внутрішнього коду, поперед якого ставиться символ # . Наприклад, #97, #65.

Множина допустимих значень - множина всіх символів (255).

Допустимі операції - операції відношень.

Функція	Призначення	Тип аргументу	Тип результату
Ord(c)	Визначає код символу C	Символьний	Цілий
Chr(n)	Визначає символ по його коду.	Цілий	Символьний
Pred(c)	Визначає символ, який знаходиться перед C	Символьний	Символьний
Succ(c)	Визначає символ, який знаходиться після C	Символьний	Символьний

Логічний тип

Ім’я типу Boolean. Значення займає 1 байт.

Область допустимих значень - true, false.

Допустимі операції - логічні операції і операції відношень.

---

2) Типи визначені користувачем

Користувач може створити свій власний тип даних. До простих типів, визначених користувачем, відносяться **перелічувальний** і **обмежений (інтервальний)** типи даних.

Перелічувальний тип даних

Цей тип означається переліченням по порядку всіх його значень. Наприклад, можна перелічити дня тижня, меблі, дерева, кольори, місяці, і т.д. Значення типу подаються іменами(ідентифікаторами), які є константами цього типу. Перелік обмежується *дужками*. Наприклад:

Дні тижня: (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)

Кольори: (red, white, black, green, blue).

Допустимими значеннями для цього типу є тільки ті значення, що приведені в описі.
Допустимими операціями є операції відношень.

Перелічені дані вважаються впорядкованими в тому порядку, в якому вони описані. Тому значення, приведене пізніше вважається більше попереднього. Кожному елементу списку привласнюється порядковий номер, починаючи з нуля.

Для аргументів перелічувального типу можна застосовувати функції *системної бібліотеки*:

Функція	Призначення	Тип аргументу	Тип результату
Ord(c)	Визначає порядковий номер значення в списку, починаючи з нуля	Перелічений	Цілий
Pred(c)	Визначає значення, яке знаходиться перед вказаним	Перелічений	Перелічений
Succ(c)	Визначає значення, яке знаходиться після вказаного	Перелічений	Перелічений

До перелічувального (порядкового) типу відносяться і деякі стандартні типи, а саме: **всі цілі типи, символьний і логічний.**

Обмежений тип даних

Обмежений тип визначає вказаний інтервал(діапазон) деякого базового перелічувального типу.

Формат опису інтервалу:

<мінімальне значення> .. <максимальне значення>

Наприклад, інтервал оцінок 2..5, інтервал днів тижня 1..7, інтервал символів ‘в’.. ‘н’, інтервал кольорів White..Blue.

Допустимими значеннями для даного типу будуть значення з вказаного інтервалу. Допустимими операціями будуть ті, що допустимі для базового типу.

Можна використовувати функції системної бібліотеки, допустимі для базового типу.

Арифметичні операції, функції, вирази. Арифметичний оператор присвоювання

ОПЕРАТОР ПРИСВОЮВАННЯ

Знак оператора **:=.**

Формат оператора:

<ім’я змінної>:=<вираз>;

Тип змінної і виразу повинні бути одного типу або бути сумісними по типу.

Наприклад:

```
Var
a,b:integer;
x,y:real;
s:string;
T:boolean; a:=24*b+4;
x:=sin(y)-7.8;
```

Оператор y:=a+b; буде вірним тому, що ціле значення можна розмістити в дійсній змінній.

А навпаки (a:=y+x;) не допустимо.

Змінній T присвоїти true, якщо a>b і false в іншому випадку: T:=a>b;

Змінній T присвоїти true, якщо a кратно b і false в іншому випадку: T:=a mod b=0;

Оператори ВВОДУ-ВИВОДУ

Оператори вводу

Оператори вводу призначені для вводу даних. Пристроєм вводу по умовчанню є клавіатура.
Формат оператору вводу:

read | readln [(<список вводу>)];

Оператор readln відрізняється від read тим, що після вводу необхідної кількості даних здійснює перехід на початок наступного рядка. Оператор readln; без параметрів використовується для затримки роботи програми. Програма продовжить роботу при натисканні довільної клавіші. Таку форму оператора часто використовують, наприклад, перед завершенням роботи програми.

Елементами списку вводу можуть бути тільки змінні. Елементи списку вводу відділяються комами.
При введенні значень з клавіатури дані можуть розділятись символами „пропуск" і Enter(¿).
Строкові дані можуть розділятись тільки Enter.
Дані, що вводяться повинні відповідати списку вводу: їх кількість повинна бути не меншою і відповідні елементи повинні бути одного типу, або сумісними по типу.
Значення, що вводяться з клавіатури, мають строковий формат, тому при введенні числових значень вони перетворюються у внутрішній формат і заносяться у відповідні змінні.

Оператор починає обробку введених даних після натискання Enter. Якщо введені всі дані з списку вводу, то оператор завершується. Якщо ні – очікується продовження вводу.

Наприклад, при наявності в програмі опису:

```
Var

a,b:integer;

x,y:real; (a)

s,s1:string[20];
```

і оператору readln(a,b,x,y);

Для задоволення вводу можна набрати на клавіатурі наступний рядок даних -34 45 3.45 -56.8 ¿
В результаті вводу змінні будуть мати значення: a=-34, b=45, x=3.45, y=-56.8.
Після виконання оператору readln(a,b,x,b); і вводу 34 45 3.45 -56 ¿
змінні будуть мати значення: a=-34, b=-56, x=3.45
Якщо треба послідовно виконати оператори readln(a,b); read (x,y);

і на клавіатурі набрати дані 22 7 2.35 -5.4 ¿

то після вводу двох перших значень курсор перейде на новий рядок (тому, що використано оператор readln) і значення для x і y введені не будуть. Тому необхідно використати наступний ввод

22 7 ¿

2.35 -5.4 ¿

Оператори виводу

Оператори виводу призначені для виводу даних. Пристроєм виводу по умовчанню є екран монітора.

Формат оператору виводу:

write | writeln [(<список виводу>)];

Оператор writeln відрізняється від write тим, що після виводу означених даних здійснює перехід на початок наступного рядка.
Елементами списку виводу можуть бути константи, змінні, вирази. Елементи списку виводу відділяються **комами**.
Числові дані перед виведенням перетворюються в строковий формат.

Наприклад, використовуючи опис (a), виведемо сповіщення:

Writeln ('При X=',X,' Y=',Y);

Елементами списку вводу є дві строкові константи і дві змінні дійсного типу. По умовчанню дійсні дані виводяться в експоненціальній формі, що незвично для користувача. Щоб дані виводились у вигляді звичному для користувача, використовується форматування даних.
Якщо необхідно перейти на наступний рядок, можна виконати оператор Writeln;

ФОРМАТУВАННЯ ДАНИХ ПРИ ВИВОДІ

Для форматованого виводу всіх типів даних необхідно вказати ширину поля виводу, тобто кількість позицій, які будуть використані для виводу значення, а для дійсних даних ще й кількість знаків після коми.

При виведенні числових цілих, символьних, логічних, строкових значень вказується тільки ширина поля, яка записується в операторі виводу після відповідного елементу виводу, відділяючись від нього символом **„:”.**

<елемент виводу>:<ширина поля виводу>

Значення ширини поля виводу може буди цілим числом або виразом цілого типу.
Дані, що виводяться вирівнюються по правому краю поля. Ліві нулі не виводяться.

Наприклад: при використанні оператора writeln (‘Значення’:15, a:5, b:7);

при a=25 | b=165 буде виведено:

_ _ _ _ _ _ _ Значення _ _ _25_ _ _ _ 165

При виведенні числових дійсних значень необхідно додатково вказати кількість знаків після крапки. Кількість вказується після ширини поля, відділяючись символом **„:”.**

<елемент виводу>:<ширина поля виводу>:<кількість знаків після крапки>

Дані, що виводяться вирівнюються по правому краю поля. Число округлюється по правилам математики з урахуванням вказаної точності.

Наприклад: при використанні оператора writeln (x:7:2, y:10:3, z:5.0);

при x=25.0034, b=-165.56784, z=356.776 буде виведено:

_ _ 25.00 _ _-165.568 _ 357.

Керування порядком обчислень. Розгалуджені алгоритми

Розглянемо алгоритмічні конструкції вибору та повтору

ОПЕРАТОРИ РЕАЛІЗАЦІЇ РОЗГАЛУДЖЕНИХ АЛГОРИТМІВ

1. Умовний оператор

Використовується для вибору дій в залежності від якоїсь умови.

Формат:

If <умова> then <оператор 1> [else <оператор 2>];

<умова> - логічний вираз

<оператор 1, 2> - може бути простим і складним

Якщо результат умови є істина, то виконується <оператор 1> , а потім управління передається оператору, що стоїть за умовним, інакше управління передається оператору, що стоїть за словом Else, якщо воно є, а потім оператору, що стоїть за умовним; якщо його нема, то зразу ж оператору, що стоїть за умовним.

<оператор 1> в свою чергу може бути умовним оператором. Такий оператор називається ***вкладеним***. Фраза Else завжди відноситься до останнього умовного оператора. Тому, якщо внутрішній умовний оператор не має фрази Else, то <оператор 1> потрібно обмежувати операторними дужками (приклад 2).

---

2. Оператор вибору

Використовується в тому разі, коли кількість варіантів вибору при розгалуженні більше двох.

Формат:

Case <перемикач> of

<набір значень 1>: <оператор 1>;

<набір значень 2>: <оператор 2>;

.

.

<набір значень n>: <оператор n>

[else <альтернативний оператор>];

end;

<перемикач> - вираз переліченого типу

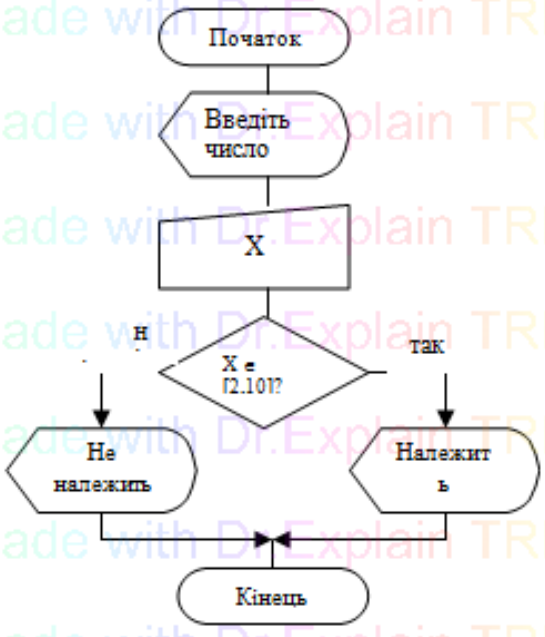
<набір значень n> - допустимі значення, які може приймати перемикач. Елементи набору відділяються комами. Значення повинні бути того ж типу, що і перемикач. Можна використовувати діапазони, наприклад 'a' .. 'm'.

<набір значень n> - оператор, що виконується для даного набору значень. Може бути простим і складним.

Алгоритм виконання: обчислюється значення перемикача, серед наборів значень розшукується обчислене значення. Якщо воно знайдене – виконується оператор, що відповідає цьому набору і управління передається оператору, що слідує за оператором вибору. Якщо значення перемикача не знайдено ні в одному з наборів даних, то виконується <альтернативний оператор>, якщо він заданий, а потім управління передається оператору, що слідує за оператором вибору. Якщо такий оператор не задано - управління передається оператору, що слідує за оператором вибору.

Розробка програм реалізації розгалужених алгоритмів

Приклад . Розробити програму визначення, чи належить задане число проміжку [2,10].



program Simbol;

```
{Визначення символів}
var
C:char;
begin
write ('Введіть символ ');
readln(C);
case C of
‘П’,’п’; writeln (‘Іванов‘);
‘І’,’і’; writeln (‘Сергій‘)
else
writeln (‘Символ не обробляється’);
end;
readln;
end.
```

ОПЕРАТОРИ ПОВТОРІВ
бувають трьох типів:

- Оператор циклу з параметром

- Оператор циклу з передумовою

- Оператор циклу з післямовою

Оператор циклу з параметром

Використовується в тому разі, коли кількість повторів оговорюється попередньо.

Формат:

For <параметр циклу>:= <початкове значення> **to | downto** <кінцеве значення> **do** <оператор>;

<параметр циклу> - змінна переліченого типу, може змінюватись на крок, що дорівнює 1 або –1. В першому випадку використовується ключове слово to, в іншому – downto.

<початкове значення> , <кінцеве значення> - вирази того ж типу, що і параметр циклу. Визначають граничні значення параметру циклу. Тип повинен співпадати з типом параметру циклу.. <оператор> - тіло циклу. Може бути простим або складним оператором.

Алгоритм виконання:

1) Визначається значення <початкове значення> , <кінцеве значення>. <параметр циклу> присвоюється значення <початкове значення>.

2) Значення <параметр циклу> зрівнюється з значенням <кінцеве значення>. Якщо значення <параметр циклу> не більше <кінцеве значення> (в разі використання фрази to) , або не менше - (в разі використання фрази downto) <кінцеве значення>, то виконується <оператор>. В протилежному випадку цикл завершується.

3) Після виконання <оператор> до <параметр циклу> додається 1 або -1, в залежності від фрази > to | downto.

4) Дії повторюються з кроку 2).

Оператор циклу з передумовою

Використовується в випадку неозначеності кількості повторів. Умова закінчення циклу перевіряється перед виконанням циклу.

Формат:

While <умова> **do** <оператор>;

<умова> - логічний вираз

<оператор> - тіло циклу. Може бути простим або складним оператором.

Алгоритм виконання:

1) Визначається значення <умова>.

2) Якщо результат істина, то виконується <оператор>. В протилежному випадку цикл завершується.

3) Після виконання <оператор> дії повторюються з кроку 1).

Таким чином, цикл повторюється до тих пір, поки значення <умова> буде істина.
Щоб оператор працював вірно, необхідно щоб в тілі циклу змінювались елементи, що входять до виразу <умова>.

Оператор циклу з післямовою

Використовується в випадку неозначеності кількості повторів. Умова закінчення циклу перевіряється після виконання циклу. Тому тіло циклу виконується хоча б один раз.

Формат:

Repeat

<оператор 1>;

.

.

<оператор n>

until <умова>;

<умова> -логічний вираз

<оператор1> . . <оператор n> - тіло циклу.

Алгоритм виконання:

1) Виконуються оператори тіла циклу.

2) Визначається значення <умова>.

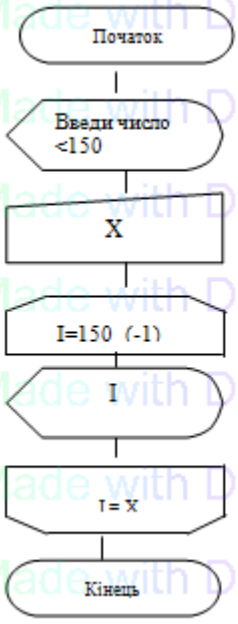
3) Якщо результат не істина, то перехід до дії 1). В протилежному випадку цикл завершується.

Таким чином, цикл повторюється до тих пір, поки значення <умова> буде не істина.

Циклічні алгоритми

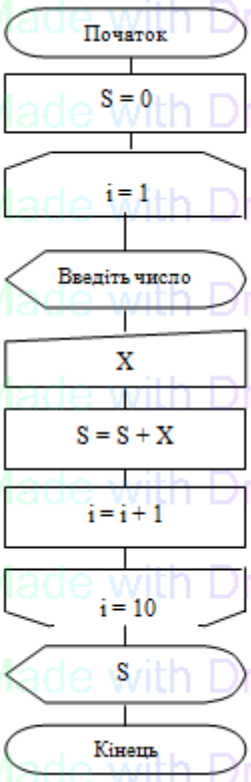
ПРОГРАММИ З ВИКОИРИСТАННЯМ ЦИКЛІЧНИХ АЛГОРИТМІВ

Приклад 1. Вивести послідовно числа від 150 до вказаного меншого числа в зворотному порядку.



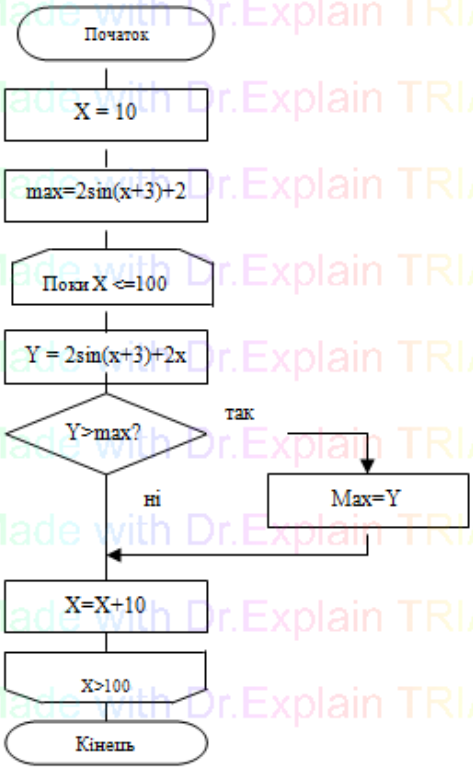

```
PROGRAM  chisla;
  var
     x,i:byte;
   {x - значення, що вводиться
   i    - змінна циклу}
   begin
     write ('введи ціле число, менше 150');
     readln(x);
     for i:=150 downto x do
         write (i:5);
  end.
```

Приклад 2. Знайти суму десяти введених значень.




```
PROGRAM  suma;
   var
     x,s:real;
     i:byte;
  {x - значення, що вводиться
   s    - сума введених значень
   i    - змінна циклу}
   begin
     s:=0;
     for i:=1 to 10 do
         begin
             write ('введи наступне значення');
             readln(x);
             s:=s+x;
         end;
     writeln ('Сума введених значень = ',s:10:2);
  end.
```

Приклад 3. Знайти найбільше значення функції Y=2sin(X+3)+2X на проміжку [10,100] в точках з кроком 10.




```
PROGRAM  Funk;
 var
   X,Y: max:real;
 begin
   X:=10;
   Max:=2*sin(X+3)+2*X;
   While X<= 100 do
     Begin
       Y:= 2*sin(X+3)+2*X;
       If Y>max then
                    Max:=Y;
       X:=X+10;
     End;
   Writeln ('Максимальне значення функції ',max:7:2);
  end.
```

Процедурно-орієнтоване програмування. Функції користувача

ПІДПРОГРАМИ

Допоміжні алгоритми - це алгоритми, які описані окремо і можуть бути використані не одноразово. Їх використання дає змогу запобігти повторних записів однакових серій операторів.
В мові Pascal **допоміжними алгоритмами (підпрограми)** можуть бути **процедури** та **функції**. У програмі процедури і функції задаються описами, які розміщуються в описовій частині.

Структура підпрограми така ж як і програми: вони складаються з заголовку і блоку.

Підпрограма повинна мати засіб для обміну даними між нею і програмою, що її викликає. Таким засобом є *формальні параметри*, які описуються в заголовку. За допомогою формальних параметрів можна передавати в підпрограму дані і отримувати їх нові значення. Параметрами можуть бути:

- *параметри-значення,*
- *параметри-змінні* (перед ними повинно стояти слово VAR),
- *параметри-процедури* (перед ними ставиться слово PROCEDURE),
- *параметри-функції* (перед ними ставиться слово FUNCTION).

Для кожного формального параметра вказується:

- ключове слово (VAR, PROCEDURE, FUNCTION), якщо воно потрібно,

- ім´я параметра,

- символ ":" та його тип.

Описи параметрів відділяються між собою символом **";"**.
Якщо програма, що викликає, повинна отримати результат, який визначається в підпрограмі, то такий параметр повинен бути описаний як параметр-змінна.

Виклик підпрограми має формат:

< ім'я підпрограми > [(< список фактичних параметрів >)]

Елементи списку відокремлюються символом ",". Між формальними та фактичними параметрами повинна бути взаємна відповідність: кількість їх повинна бути однакова, однаковий порядок слідування, тип кожного фактичного параметру повинен співпадати з типом відповідного формального параметру.

Глобальні та локальні описи

Глобальні константи, типи, змінні - це ті, що оголошені в програмі зовні процедур та функцій до їх означення. Глобальні елементи доступні всім процедурам і функціям.
Локальні - існують тільки в тілі процедури або функції, де вони визначені. Формальні параметри також є локальними.
Якщо ім´я локального опису співпадає з ім´ям глобального опису, то така змінна розглядається як локальна.

Упереджуючий опис підпрограм

При описі процедур і функцій необхідно слідкувати, щоб опис підпрограми, що викликається, був раніше виклику. Щоб не слідкувати за послідовністю записів підпрограм можна попередньо описати заголовки всіх підпрограм з словом FORWARD, а потім в довільному порядку записати опис всіх підпрограм.

Формат попереднього опису заголовку:

<заголовок підпрограми >; FORWARD;

ПРОЦЕДУРИ ТА ФУНКЦІЇ

Функції
Функції використовуються в тому разі, коли підпрограма повертає в якості результату одне значення простого типу.

Формат заголовку функції:

FUNCTION <ім´я функції> [(<список формальних параметрів>)]:

<тип результату>;

Результат функції повертається в програму, що її викликає, тільки через ім´я функції. Тому в розділі операторів функції обов´язково повинен бути оператор присвоєння імені функції значення результату або оператор присвоєння системній змінній RESULT значення результату.

Звертання до функції використовується в виразах.

Процедури
Процедури використовуються в тому разі, коли підпрограма повертає в якості результату значення структурованого типу або кілька значень, або значень не повертає..

Формат заголовку процедури:

PROCEDURE <ім´я процедури> [(<список формальних параметрів>)];

Результат функції повертається в програму, що викликала дану, за допомогою параметрів, які обов´язково повинні бути описані як параметри-змінні.

Передача в підпрограму структурованих типів даних
Всі формальні структуровані параметри повинні бути описані як параметри-змінні. Передачу в підпрограму структурованих типів даних можна здійснити декількома способами:

1. Опис свого типу даних.
В основній програмі в розділі опису типів описується структурований тип. Змінні в основній програмі і відповідний формальний параметр в заголовку підпрограми повинні бути змінними цього типу.

Type

тm = array [1....10] of real;

```
var
mas: тм;
procedure proc (var mass: тm); {заголовок п/п}
:
:
end;
begin
:
Proc (mas); {Виклик п/п}
:

end.
```

2. Використання безтипових змінних

У заголовку підпрограми не вказується тип параметру-змінної. В підпрограмі в розділі опису типів описується відповідний тип до якого буде приведено безтиповий параметр. Цей тип повинен бути сумісним з типом відповідного переданого параметру.

```
Var
b1: array [1. .100] of byte;
b2: array [‘a’. .’z’] of byte;
procedure pr (var b);
type
Tb = array [1....32500] of byte;
:
:
{Звернення до масиву b в п/п: TB(b)[I]}
Tb(b)[i]:=34;
```

```
End;

Begin

:

pr(b1);

:

end.
```

3. Масиви і рядки невизначеної довжини

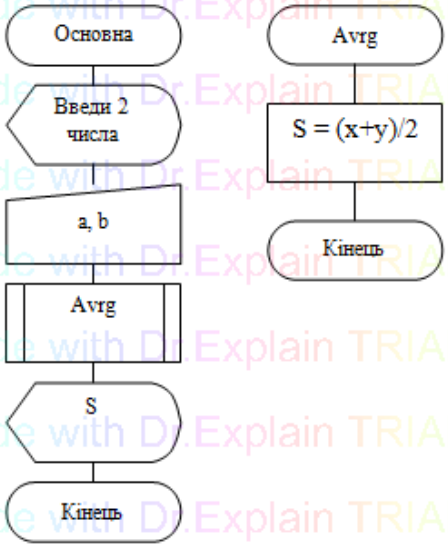
В заголовку підпрограми при описі параметру вказується його тип але не вказується його розмір. При використанні масивів значення розміру масиву можна отримати за допомогою функції High(<Ім'я масиву>). Нумерація елементів в масиві буде починатись з 0 (нуля) і закінчуватись значенням функції High(<Ім'я масиву>)-1

```
Function Sum(var X: array of Real): Real;

Var

I: Word;

S: Real;

Begin

S := 0;

For I := 0 to High(X)-1 do

S := S + X[I];

Sum := S;

End;
```

Використання підпрограм

Приклад 1. Для трьох введених цілих чисел знайти їх середнє арифметичне. Знаходження середнього арифметичного оформити у вигляді процедури.

В процедурі повинно визначатись значення і повертатись в програму, що викликала процедуру. Це значення повертається за допомогою формального параметру - змінної.

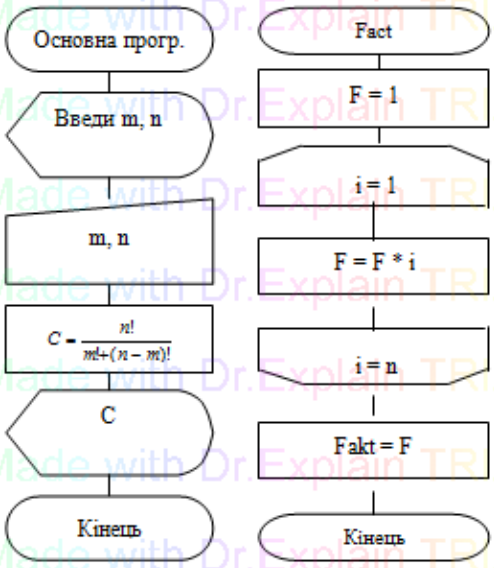



```
Program PAR;
  Var
    a,b: integer;
    S: real;
  Procedure avrg(x,y:integer; VAR S:real);
    Begin
      S:= (x+y)/2;
    End;
  Begin
    write ('введи два числа');
    readln (a,b);
    avrg (a,b,s);
    writeln ('середнє арифметичне чисел',
                  a:5,b:5,'дорівнює', S:10:2);
  End.
```

Приклад 2. Знайти кількість сполучень з використанням функції:
Сполучення обчислюються за формулою

$$n!/(m!+(n-m)!)$$

Очевидно, що визначення факторіалу повторюється, тому його визначення необхідно винести в підпрограму, а саме функцію тому, що результатом буде одне просте значення.




```
program komb;
 var
   c,n,m: word;
 function fakt (n: word): word;
   var
     i: word; f:longint;
   begin
     f:= 1;
     for i:= 1 to n do
       f:= f*i;
     fakt:= f;
   end;

begin
  writeln ('введи m,n');
  readln (m,n);
  c:= fakt (n) div (fakt(m) + fakt(n-m));
  writeln ('кількість сполучень ',n:3,'по',
                m:3,'=',c:7);
end.
```

Рекурсія

ПОНЯТТЯ РЕКУРСІЇ

Поняття підпрограми тісно пов’язане з одним методом розв’язання задач, яке має назву "рекурсія".
Рекурсія – це метод визначення чи вираження функції, процедури, мовної конструкції чи рішення задачі за допомогою тієї ж функції, процедури та т.п. Слово рекурсивний, рекурентний вийшло від латинського “recurro” (бігти назад, вертатися).

Якщо відомо алгоритм розв'язку задачі для найпростіших даних, та як звести розв’язок до більш простих даних в інших випадках, має сенс використати рекурсію.

Співвідношення, в яких для обчислення поточного значення використовуються значення, отримані на попередніх етапах обчислення називають рекурентними.

Розглянемо приклад використання рекурсії для визначення факторіалу.
За визначенням факторіалу відомо, що:

N! = 1*2*3*…*(N-1)*N.

Але вираз 1*2*3*…*(N-1) в свою чергу є (N-1)!. Тоді можна записати

N! =(N-1)! * N.

Такий вираз є рекурентним співвідношенням. З математики відомо, що 0!=1. Останній вираз називають початковим значенням.

Подивимося, яку роботу треба виконати, щоб обчислити F(N), для довільного натурального числа N. Для обчислення F(N) треба провести рекурсивне звертання до F(N-1), це в свою чергу потребує друге рекурсивне звертання до F(N-2), і т.д. Таким чином треба провести N рекурсивних звертань, останнє з яких F(0)=1. Кількість рекурсивних звертань називають глибиною рекурсії. Для F(N) глибина рекурсії дорівнює N.

Конструкції мови Pascal дозволяють використовувати рекурсію. Так, для визначення факторіалу можна записати таку функцію:

```
function f(n:integer): longint;

begin

if n=0 then f=1

else f=n*f(n-1);

end.
```

Приклад 1 . Визначити N-й елемент послідовності чисел Фібоначі: 1, 1, 2, 3, 5, 8, 13, …

Ця послідовність названа на ім’я італьянського математика Фібоначчі, який в 1202 році сформулював таку задачу:

Кожний місяць самка з пари кролів приносить двох кроликів (самця та самку). Через два місяці нова самка сама приносить пару кроликів. Треба знайти кількість кролів в кінці року, якщо на початку року була одна новонароджена пара кролів і в продовж цього року кролі не вмирали.

Ряди Фібоначчі завжди грали велику роль в математиці. Якщо послідовно ділити два поточних останніх членів ряду один на другий, то отримаємо результат, який відомо під назвою “золотий перетин”. В архітектурі цей результат подається як класична старовина, в ньому признають тайну силу.

N-й елемент цієї послідовності дорівнює сумі двох елементів, які безпосередньо передують йому:

F(N)= F(N-1) + F(N-2).

Початкове значення F(1)=1; F(2)=1.

Функція обчислення чисел Фібоначчі на мові Pascal:

```
function fib(n:integer): integer;

begin

if (n=1) or (n=2) then fib=1

else fib=fib(n-1) + fib(n-2);

end.
```

ОДНОВИМІРНІ МАСИВИ

Масив - це обмежена послідовність елементів одного типу.
Кожний елемент в масиві має свій порядковий номер. Елементи масиву можна перерахувати (проіндексувати). Індексом може бути довільний перелічувальний тип.

Масив характеризується іменем, розміром і типом елементів.

Масиви можуть бути **одновимірні** і **багатовимірні**.

Одновимірні масиви

Формат опису типу "масив" для одновимірних масивів:

```
type
<ім’я типу> = ARRAY <[<диапазон індексу>]> OF <тип елементів>;
```

<диапазон індексу> - діапазон довільного перелічувального типу. Якщо діапазон охоплює всі елементи типу, то замість діапазону можна вказувати ім’я типу, наприклад, BYTE, CHAR.

Наприклад:

```
type
    Tmas = array [1..10] of real;
Tmas1= array [10..100] of char;
    Tmas2= array [‘a’..’k’] of integer;
    Tmas3= array [byte] of string[30];
```

Для використання масиву в програмі необхідно об’явити змінні типу „масив”:

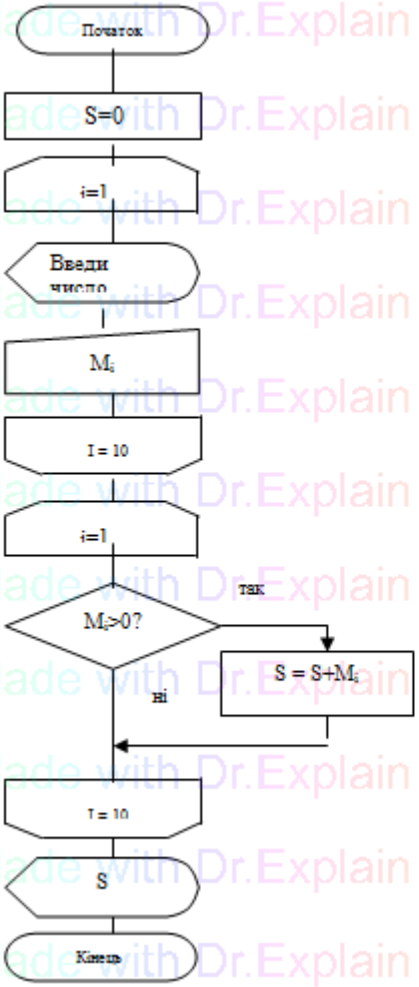
```
Var
    mas: Tmas;
    mas1: Tmas1;
    mas2: Tmas2;
    mas3: Tmas3;
    mas4: array [1..10] of word;
```

Кожний елемент масиву характеризується іменем масиву і індексом. Наприклад, mas1[11], mas3[2]. Такі змінні називають індексованими змінними.
Область допустимих значень для масиву визначається типом елементів масиву.
Операції над масивами: не визначені, тобто не допускаються ніякі операції. Але для однотипних масивів можна виконувати оператор присвоєння. Наприклад, якщо маємо опис

```
Var
    mas5,mas6: Tmas;
```

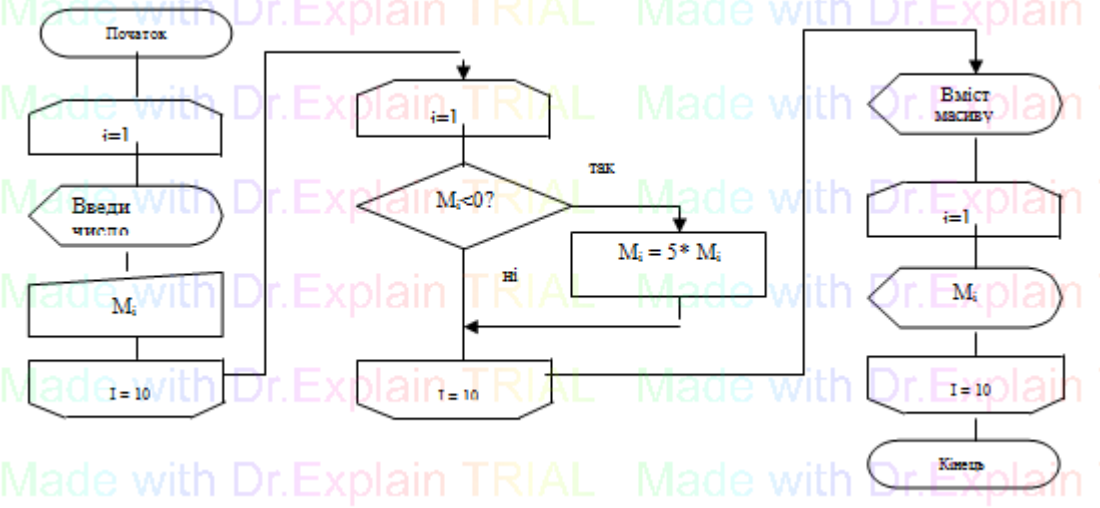
то можна виконати mas5:=mas6; В результаті вміст масиву mas5 буде таким же, як і вміст масиву mas6.
Щоб обробити масив(ввести, вивести, змінити), необхідно звернутись до кожного елементу масиву, тобто, для обробки масивів використовуються оператори циклу.

Приклад 1. Для заданого масиву з 10 дійсних значень знайти суму позитивних значень.




```
PROGRAM  suma;
{Знаходження суми позитивних значень в масиві
з 10 дійсних чисел.}
   var
      M:array [1..10] of real;
      s:real;
      i:byte;
  {M    - масив значенньь
   s    - сума введених значень
   i    - змінна циклу}
  begin
      s:=0;
      for i:=1 to 10 do
          begin
              write ('введи число');
              readln(M[i]);
          end;
      for i:=1 to 10 do
         if M[i]>0 then
                        s:=s+ M[i];
      writeln ('S = ',s:10:2);
  end.
```

Приклад 2. В заданому масиві з 10 дійсних значень кожне негативне значення зменшити в разів.



PROGRAM masiv;

{Заміна негативних позитивних значень в масиві з 10 дійсних чисел на значення в 5 разів збільшене.}

var

M:array [1..10] of real;

i:byte;

{M – масив значень

i - змінна циклу}

begin

for i:=1 to 10 do

begin

write ('введи число');

readln(M[i]);

end;

for i:=1 to 10 do

```
if M[i]< 0 then
M[i]:=5*M[i];
writeln ('Новий вміст масиву’);
for i:=1 to 10 do
write (M[i]);
writeln;
end.
```

БАГАТОВИМІРНІ МАСИВИ

Для багатовимірних масивів вказуються діапазони для кожного індексу, перелічені через кому. Індексом може бути будь-який перелічений тип.
Найчастіше використовуються **двовимірні масиві**, які називають **матрицями**.

Формат опису типу "масив" для двовимірних масивів:

```
type
<ім’я типу> = ARRAY <[<диапазон індексу 1>,<диапазон індексу 2>]> OF
<тип елементів>;
```

Наприклад:

```
type
Tmatr= array [1..10,1..10] of word;
Tmatr1= array ['a'..'z',1..100] of integer;

Var
    matr: Tmatr;
    matr1: Tmatr1;
```

Елемент матриці характеризується іменем матриці і двома індексами. Наприклад, matr[2,3].

Деякі відомості про матриці:

Матриця має вигляд

a11 a12 a13

a21 a22 a23

a31 a32 a33

Матриця складається з рядків і стовпців. Перший індекс вказує на номер рядка, а другий - на номер стовпця. Розмір матриці визначається добутком кількості рядків на кількість стовпців.
Якщо кількість рядків дорівнює кількості стовпців, то матриця називається *квадратною*.

Елементи матриці, які мають однакові індекси, складають головну діагональ матриці. В нашому прикладі це елементи (a11, a22, a33). Елементи, що лежать над головною діагоналлю - (a12, a13, a23). Можна помітити, що у цих елементів другий індекс більше першого. Елементи, що лежать під головною діагоналлю - (a21, a31, a32). У цих елементів другий індекс менше першого.

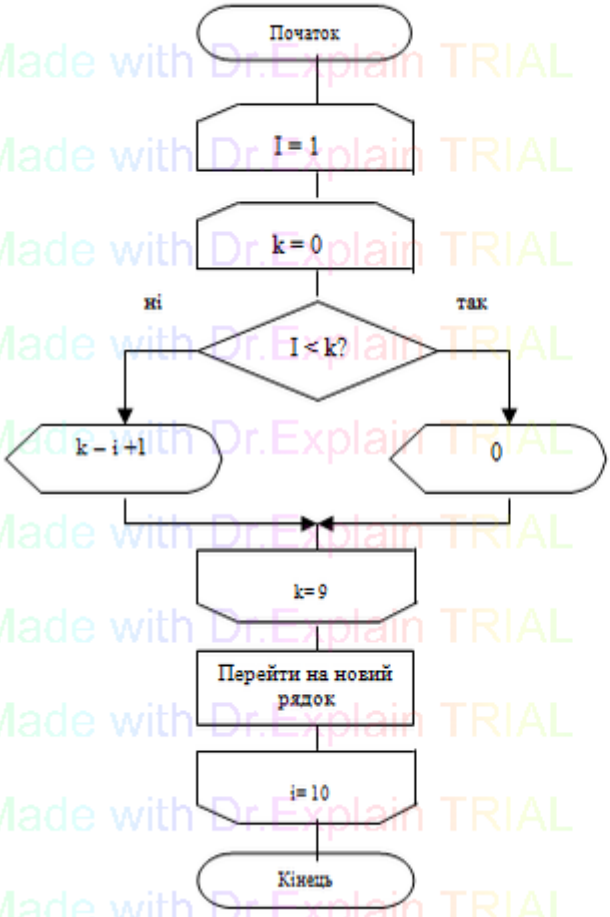
Матрицю можна розглядати як масив рядків, де кожний рядок є масив. Тобто опис

mas= array [1..10,1..10] of word; і опис

mas2= array [1..10] of array [1..10] of word; є ідентичними.

Можна звертатись до рядка, як до масиву, наприклад, mas[3].

Приклад 1. Вивести на екран матрицю у вигляді



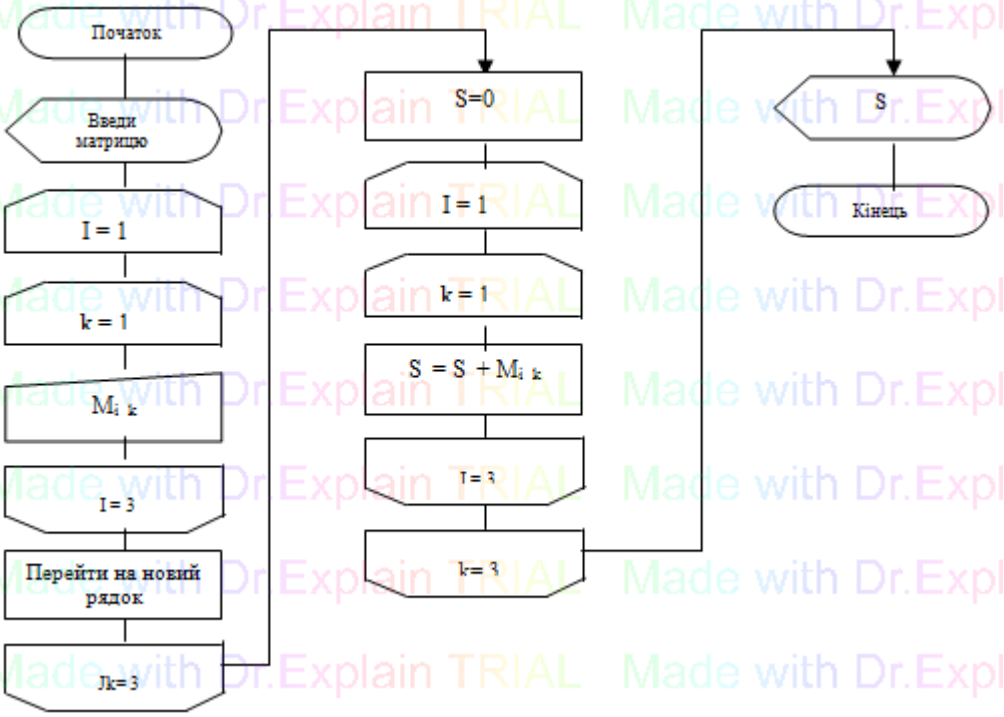

```
1000000000
2100000000
3210000000
4321000000
..........
10987654321
```

```
program DownAngle;
var
  i,k:word;
begin
  for i:=1 to 10 do
    begin
      for k:=0 to 9 do
        if  i<k then
                    write ('0':3)
                  else
                    write(i-k:3) ;
      writeln;
    end;
  readln;
end.
```

Результати виконання

```
1 0 0 0 0 0 0 0 0 0
2 1 0 0 0 0 0 0 0 0
3 2 1 0 0 0 0 0 0 0
4 3 2 1 0 0 0 0 0 0
5 4 3 2 1 0 0 0 0 0
6 5 4 3 2 1 0 0 0 0
7 6 5 4 3 2 1 0 0 0
8 7 6 5 4 3 2 1 0 0
9 8 7 6 5 4 3 2 1 0
10 9 8 7 6 5 4 3 2 1
```

Приклад 2. Знайти суму елементів квадратної матриці 3*3.



Програма:

```
program suma;
    var
     matr:array[1..3,1..3] of real;
     i,k:word;
     s:real;
    begin
     writeln (‘ Введіть матрицю у вигляді матриці’);
     for i:=1 to 3 do
       begin
for k:=1 to 3 do
          read (matr[I,k];
        readln;
       end;
     s:=0;
```

```
  for i:=1 to 3 do
      for k:=1 to 3 do
          s:=s+matr[i,k];
  writeln ('Сума елементів матриці=',s:6:2);
end.
```

Рядки

Робота з строковим типом даних

Тип строкових даних (рядок) оголошується як STRING;
Змінні типу STRING можуть набувати значень змінної довжини. Максимальна кількість символів у рядку дорівнює **256**.
Якщо завчасно відомо, що кількість символів не може бути більше якогось значення N, то рядок можна обмежити до N символів, оголосивши: STRING [N], де N- ціле невід'ємне значення від 1 до 256.

Строкову змінну можна розглядати, як масив символів., тобто

a: string[10]; можна інтерпретувати, як a: array[1...10] of char;

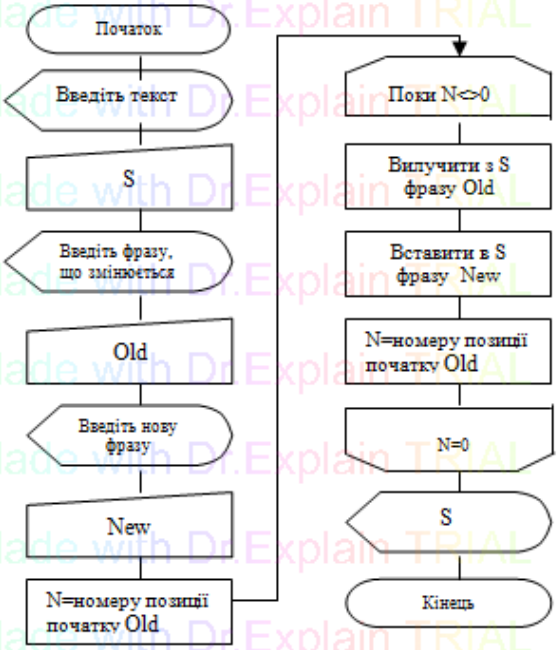
Для вводу значень змінних типу string використовується оператор readln, а не read. Одним оператором можна ввести тільки один рядок.
Пустий рядок визначається як два апострофи, записані поряд: ‘’

Стандартні функції і процедури для роботи з рядками.

Функція, процедура	Призначення	Приклад
Length(S)	Функція. Визначення довжини рядка.	length (‘’)=0; length (‘авто’)=4.
concat (s1,s2,s3)	Функція. Об'єднання рядків.	concat (‘транс’,’фор’,’матор’)=’трансформатор’
pos (<підрядок>,<рядок>)	Визначення позиції входження підрядка в рядок.	pos (‘и’,’инструкция’)=1;
Copy(S,<номер символу початку>, <кількість символів>)	Функція. Коріювання підрядка	copy (‘информатика’,1,4)=’инфо’
delete (S,<номер символу початку>, <кількість символів>)	Видалення підрядка.	Виконавши delete (‘автомобиль’,5,6) отримаємо ‘авто’
insert (<підрядок>,<рядок>,<номер символу початку вставки >)	Вставка підрядка	виконавши insert (‘дорога’, ‘Добрий день, Олю’,13) отримаємо ‘Добрий день, дорога Олю !’
str (<число>,<змінна рядку>)	Переведення числового значення в рядок.	для z: integer=1888, line : string; Після str(z,line) отримаємо line=’1888’
Val (<число в строковому форматі>, <числова змінна>, <С – код завершення>)	Переведення числа в строковому форматі в число. Якщо при перетворенні помилку не виявлено, то С = 0, інакше С набуває значення номера позиції помилкового символу	для line : string=188.72; x : real; c : word; Після var (line,x,c) отримаємо x=188.72; c=0; Якщо line = ‘2dr5’ після val(lim,x,c) x-невизначені, c=2.

Розробка програм з використанням строкових даних.

Приклад 1. В заданому тексті замінити всі входження заданого слова на інше задане слово.




```
program tekst;
var
  S,old,new:string;
  n:integer;
begin
  writeln('Введи текст');
  readln(S);
  writeln('Введи фразу, що змінюється');
  readln(old);
  writeln('Введи фразу для заміни');
  readln(new);
  n:=pos(old,S);
  while n<>0 do
    begin
      delete(S,n,length(old));
      insert(new,S,n);
      n:=pos(old,S);
    end;
  writeln(S);
end.
```

Навчальні ресурси

1. Онлайн посібник мови Pascal: http://pascal.org.ua/
2. Інформатика і програмування: http://mojainformatika.ru/
3. http://www.abc-it.lv/index.php/id/751
4. http://weboap.kspu.edu/