МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФІЗИКО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

І. М. Ломонос, Л. І. Ванжа Н. А. Мацецка

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з курсу
ОСНОВИ АЛГОРИТМІЧНИХ МОВ

Вінниця
ДонНУ
2019

УДК 004.423.2(072)
Л 753

Рекомендовано до друку вченою радою
фізико-технічного факультету ДонНУ імені Василя Стуса
(протокол № 4 від 25.11.2019 р.)

Автори: *І. М. Ломонос,* завідувач навчально-наукової лабораторії програмного забезпечення систем штучного інтелекту кафедри комп'ютерних технологій фізико-технічного факультету ДонНУ імені Василя Стуса;
Л. І. Ванжа, завідувач навчально-наукової лабораторії спеціального програмного забезпечення кафедри комп'ютерних технологій фізико-технічного факультету ДонНУ імені Василя Стуса.
Н. А. Мацецка, старший викладач кафедри комп'ютерних технологій фізико-технічного факультету ДонНУ імені Василя Стуса;

Рецензент: *С. Є. Фурса,* канд. техн. наук, доцент, в. о. завідувача кафедри комп'ютерних технологій фізико-технічного факультету Донецького національного університету імені Василя Стуса.

Л 753 Основи алгоритмічних мов: методичні вказівки / І. М. Ломонос, Л. І. Ванжа, Н. А. Мацецка. Вінниця: ДонНУ імені Василя Стуса, 2019. 96 с.

Методичні вказівки містять навчальні матеріали з курсу «Основи алгоритмізації та програмування» та призначені для студентів денної форми навчання студентів фізико-технічного факультету.

Видання вміщує:

- рекомендації до виконання лабораторних робіт, вимоги щодо оформлення та захисту лабораторних робіт, перелік теоретичних питань, що розглядаються в межах курсу, а також список рекомендованої літератури;
- довідкову інформацію щодо концепції типів даних, методів організації обчислювального процесу, а також короткий довідник стандартних функцій та директив компілятора середовища розробки Turbo Pascal 7.0 та помилок періоду виконання програми;
- деякі базові шаблони програмування.

У теоретичній частині методичних вказівок викладаються основи структурного програмування, базова концепція типів даних та основні алгоритми обробки даних. Практична частина курсу складається з практики самостійної розробки програм мовою високого рівня Pascal у середовищі розробки Turbo Pascal 7.0.

Курс базується на знаннях студентів з предмету «Інформатика» середньої школи.

УДК 004.423.2(072)

ЗМІСТ

Розділ 1. Методичні вказівки щодо виконання лабораторних робіт

Вимоги до оформлення звіту з лабораторної роботи та її захисту

Звіт має бути виконаний на аркушах формату А4 та складатися з таких частин:

1. *Титульний аркуш* із зазначенням номера лабораторної роботи, назви курсу, ПІБ студента, курсу та групи студента.

2. *Аркуш завдання* потрібно отримати в електронному вигляді у викладача. Аркуш завдання являє собою список контрольних питань та індивідуальних завдань щодо певного варіанта та є витягом з лабораторного практикуму.

3. *Відповіді на контрольні питання.* Відповідь на кожне контрольне питання повинна бути лаконічною, повною та однозначною. При захисті лабораторної роботи, а також на момент модульного та екзаменаційного контролю студент повинен бути готовий пояснити смисл відповіді на питання в усній формі.

4. Для кожного пункту індивідуального завдання пропонується *лістинг*, що включає: текст програми із зазначенням прізвища студента, номера індивідуального завдання та тексту вирішеного завдання; результатів тестування програми. Результати тестування програми надаються у вигляді трьох груп тестів для різних вхідних даних: допустимі, недопустимі та в умовах меж.

Звіт може бути виконано українською мовою письмово або надруковано. Захист звіту проводиться тільки у межах лабораторних занять певної групи студентів або спеціального додаткового часу (що може бути виділено за певних навчальних обставин) та виконується у три етапи:

1. Надання теоретичної частини звіту (пункти 1–3) викладачу, виправлення помилок у теоретичній частині. Теоретична частина вважається захищеною, коли вона не містить помилок, та студент здатний правильно відповісти на кожне контрольне запитання усно без допомоги. Після захисту теоретичної частини на звіті ставиться позначка про успішний захист теоретичної частини та дату захисту.

2. Демонстрація індивідуальних завдань на комп’ютері. За умови правильності роботи програми ставиться позначка про демонстрацію програми на аркуші завдання із зазначенням дати демонстрації. *Важливо!* Ця позначка не додає балів до лабораторної роботи, але є обов’язковою для захисту.

3. Захист лістингів з індивідуальних завдань. Лістинг вважається захищеним, якщо студент продемонстрував роботу програми та може відповісти про значення будь-якого фрагменту коду програми.

Лабораторна робота вважається захищеною, якщо на титульному аркуші є позначка про захист із зазначенням кількості набраних балів, дати захисту та підписом викладача або асистента, що прийняв захист.

Лабораторна робота обов’язково повинна мати відповіді на контрольні запитання, але може не містити всіх пунктів індивідуального завдання, а лише декілька. В такому випадку за лабораторну роботу надається кількість балів, що менша на кількість невиконаних завдань.

Студент може розраховувати на максимальну кількість балів лише у випадку захисту лабораторної роботи *до поточного модульного контролю відповідного модуля*. У випадку захисту лабораторної роботи наступного модуля або на додаткових зайняттях, за лабораторну роботу може бути отримано лише до 1/3 від максимальної кількості балів.

Лабораторні роботи студент повинен зберігати до кінця навчального курсу та мати їх при собі під час модульного та екзаменаційного контролю.

У розділі подані завдання для лабораторних робіт з прикладом завдання за варіантом. Повний перелік завдань за варіантом знаходиться в лабораторному практикумі з курсу «Основи алгоритмізації та програмування». Завдання за варіантом змінюються кожного року.

У частині «Перелік теоретичних питань» перераховані теоретичні питання, що розглядаються під час вивчення навчального курсу «Основи алгоритмізації та програмування». Перелік поділений, згідно з навчальними модулями, на контроль кожного модуля виносяться питання, що були розглянуті в поточному модулі. На екзамен виносяться питання з обох модулів семестру. Кожне питання розглядається на лекційних заняттях та закріплюється практично під час виконання та захисту лабораторних робіт. Проте глибоке опанування матеріалу передбачає самостійну роботу студента – не менш ніж 5 годин самостійної роботи на кожні 2 години (одного заняття) лекційного матеріалу.

Виконання лабораторних робіт передбачає не тільки самостійну роботу студента, але й роботу в колективі під час лабораторних занять.

Перелік теоретичних питань

Модуль 1

1. Етапи підготовки та розв’язання задач на комп’ютері.
2. Парадигми та мови програмування.
3. Поняття алгоритму, властивості та засоби запису.
4. Життєвий цикл програми.
5. Середовище розробки програмного забезпечення.
6. Поняття компіляції та відлагодження програми.
7. Структура програми.
8. Лінійний обчислювальний процес.
9. Загальна концепція типів мови Pascal.

10. Представлення даних в пам’яті обчислювальної машини.
11. Базові порядкові типи даних (мети, символьний, логічній): представлення в пам’яті та інтерпретування.
12. Поняття змінної та ініціалізації.
13. Операція присвоєння.

Модуль 2

14. Дійсні типи даних: представлення в пам’яті та інтерпретування.
15. Операції введення / виведення.
16. Правила виклику підпрограм.
17. Рядковий тип даних: представлення в пам’яті та інтерпретування.
18. Операції та вирази.
19. Явне та неявне перетворення типів.
20. Розгалужений обчислювальний процес.
21. Циклічний обчислювальний процес.

Модуль 3

22. Структурований тип даних «масив»: визначення, представлення в пам’яті та інтерпретування.
23. Алгоритми пошуку.
24. Алгоритми впорядкування.
25. Засоби організації підпрограм.
26. Рекурсивний обчислювальний процес.
27. Перерахунковий тип даних: визначення, представлення в пам’яті та інтерпретування.
28. Тип даних запис: визначення, представлення в пам’яті та інтерпретування.
29. Файлові типи даних.

Модуль 4

30. Вказівники та адреси.
31. Робота з динамічною областю пам’яті.
32. Поняття модульності та повторного використання в програмуванні.
33. Стандартні модулі.
34. Організація модулів.
35. Поняття списку. Односпрямований та двонаправлений списки. Методи організації та обробки.
36. Організація, принципи обробки та методи створення фундаментальних структур даних (колекція, стек, черга, дерево).
37. Поняття об’єктно-орієнтованого програмування.

Рекомендована література

1. Немнюгин С. А. Turbo Pascal. СПб.: Издательство «Питер», 2000. 496 с.
2. Шень. А. Программирование: Теоремы и задачи. М.: МЦНМО, 2004. 149 с.
3. Фаронов В. В. Турбо Паскаль 7.0. Начальный курс: учебное пособие. М.: Издательство «ОМД Групп», 2003. 616 с.
4. Фаронов В. В. Turbo Pascal. СПб.: БХВ-Петербург, 2004. 1 056 с.
5. Шпак Ю. А. Turbo Pascal 7.0 на примерах / под ред. Е. С. Ковтанюка. К.: Издательство Юниор, 2003. 496 с.
6. Тишин В. И. Паскаль. Основы программирования. Программирование в интегрированных средах Turbo Pascal 7, Borland Pascal for Windows, 2002. Т. 1. 424 с. Т. 2. 380 с.
7. Марченко А. И., Марченко Л. А. Программирование в среде Turbo Pascal 7.0 / под ред. В. П. Тарасенко. К.: ВЕК+, 2000. 464 с.
8. Кормен Т. Х., Лейзерсон Ч. И. и др. Алгоритмы. Построение и анализ. СПб.: Вильямс, 2008. 1 296 с.
9. Лавров С. С. Программирование. Математические основы, средства, теория. СПб.: БХВ-Петербург, 2001. 320 с.
10. Окулов С., Пестов А., Пестов О. Информатика в задачах Киров: ВГПУ, 1998.
11. Верещагин Н. К., Шень А. Лекции по математической логике и теории алгоритмов. Части 1–3. М.: МЦНМО, 1999.
12. Кнут Д. Искусство программирования для ЭВМ. Т. 1–3. СПб.: Вильямс, 2000.
13. Вирт. Н. Алгоритмы и структуры данных. М.: Мир, 1989.
14. Бауэр Ф. Л., Гооз Г. Информатика. Часть 1, 2. М.: Мир, 1990.
15. Бондарев В. М., Рублинецкий В. И., Качко Е. Г. Основы программирования. Харьков: Фолио; Ростов н/Д: Феникс, 1998. 368 с.
16. Пильщиков В. Н. Сборник упражнений по языку Паскаль. М.: Наука, 1989.
17. Бардачов Ю. М., Костін В. О., Ходаков В. Є. Тлумачний російсько-український-англійський словник. Олді-плюс, 2006.

₪ Лабораторна робота № 1

- **Тема:** Середовище Turbo Pascal. Синтаксис мови Pascal. Структура програми. Лінійні алгоритми.
- **Мета:** ознайомлення з середовищем Turbo Pascal; вивчення структури програми на прикладі лінійного алгоритму; вивчення засобів створення та виконання програми.

 **Контрольні питання** 

1. Опішить зовнішній вигляд екрана та розташування елементів інтерфейсу користувача при роботі з середовищем Turbo Pascal.
2. Дайте визначення терміна «компіляція програми» та призначення компілятора як компоненти середовища Turbo Pascal.
3. Дайте визначення терміна «компоновка програми» та призначення компонувальника як компоненти середовища Turbo Pascal.
4. Дайте визначення терміна «відлагодження» та призначення програми налагоджувальника як компоненти середовища Turbo Pascal.
5. Яке призначення має вікно буфера обміну (Clipboard Window)?
6. Опішить призначення комбінацій клавіш:
   I. Ctrl+Ins;
   II. Ctrl+Del;
   III. Shift+Ins;
   IV. Shift+Del.
7. Опішить призначення комбінацій клавіш:
   I. F7;
   II. F8;
   III. F4;
   IV. Ctrl+F2;
   V. Ctrl+F4;
   VI. Ctrl+F7;
   VII. Ctrl+F8;
   VIII. Ctrl+F9.
8. Надайте комбінації клавіш для зміни активного вікна.
9. Як відмінити останню дію в текстовому редакторі Turbo Pascal?
10. Надайте комбінації клавіш для таких дій:
    I. Компіляція програми;
    II. Компіляція та компоновка програми;
    III. Запуск програми з середовища;
    IV. Аварійне зупинення програми.

11. Надайте визначення терміна «точка переривання» (breakpoint). Як встановити умовну та безумовну точку переривання в середовищі Turbo Pascal?
12. Опішить структуру програми мовою Pascal.

 **Індивідуальне завдання** 

1. Наберіть в редакторі задану програму (Додаток 3). Виправте помилки та виконайте її.

Оцінюються навички роботи з середовищем Turbo Pascal, кількість зроблених додаткових помилок при наборі програми та підхід до їхнього виправлення тощо. Не забувайте, що зберігати програму необхідно в своєму каталозі.

2. Виконайте програму в покроковому режимі. Спостерігайте за станом змінних. У процесі виконання програми змініть значення однієї зі змінних та вирахуйте, чому дорівнює помноження змінних A та B.

Оцінюється впевненість студента при відповіді на питання викладача та повнота відповіді. Студент повинен вміти виконувати програму в покроковому режимі, стежити за станом змінних, змінювати їхні значення в процесі виконання програми.

3. Встановіть умовну та безумовну точки переривання програми. Виконайте програму. Проаналізуйте поведінку середовища. Зніміть точки переривання. Виконайте програму використовуючи динамічну точку переривання.

Оцінюється впевненість студента при відповіді на питання викладача та повнота відповіді. Студент повинен вміти користуватися умовними, безумовними точками переривання: встановлення, видалення, редагування.

4. Опішить налаштування середовища Turbo Pascal (пункти меню Options та Window).

Оцінюється впевненість студента при відповіді на питання викладача та повнота відповіді. Студент повинен розуміти призначення налаштувань середовища Turbo Pascal.

₪ Лабораторна робота № 2

- **Тема:** Тип даних, змінна та константа. Базові та похідні типи даних.
- **Мета:** ознайомлення з представленням даних в мові Pascal; вивчення концепції типів даних мови Pascal; отримання навичок у визначенні типу даних, змінних та констант; використання операції присвоєння; опанування системи допомоги середовища Turbo Pascal.

Контрольні питання

1. Опишіть комбінації клавіш при роботі з системою допомоги середовища Turbo Pascal.
2. Надайте схему розташування даних в пам'яті для базових типів даних.
3. Надайте формальний опис визначення наступних типів даних:
   I. Масив;
   II. Запис;
   III. Множина.
4. Надайте спосіб запису та мету використання перерахункового типу даних.
5. Надайте спосіб запису та мету використання діапазонного типу даних.
6. Наведіть спосіб визначення констант базових та структурованих типів даних.

Індивідуальне завдання

1. Наберіть в редакторі задану програму (Додаток 3). Виправте помилки та виконайте її. Поясніть те, що було виведено на екран.

Оцінюється впевненість студента при відповіді на питання викладача та повнота відповіді. Студент повинен володіти знаннями щодо визначення констант різних типів даних.

2. Наберіть в редакторі задану програму (Додаток 3). Змініть її у такий спосіб, щоб вона виводила дані, які зазначено для Вашого варіанта.

Оцінюється підхід до вирішення задачі та впевненість при відповідях на додаткові питання.

№	На екрані
X	1 2 3

3. Напишіть програму, яка виведе на екран межі всіх базових порядкових типів даних. Вхідні дані – немає; вихідні дані – назва типу даних, нижня та верхня границя діапазону.

Оцінюється підхід до вирішення задачі та впевненість при відповідях на додаткові питання.

4. Напишіть програму, яка виведе на екран результат обчислення формули за варіантом. Вхідні дані – аргументи функції; вихідні дані – значення функції від введених аргументів.

Оцінюється підхід до вирішення задачі та доцільність використання тих чи інших типів даних.

№	**Формула**
X	$f = b^3 - a^{-2}$

₪ Лабораторна робота № 3

- **Тема:** Організація введення–виведення засобами мови Pascal.
- **Мета:** здобуття навичок організації введення–виведення засобами мови Pascal; вивчення шаблонів спілкування з користувачем та алгоритмів перетворення даних; організація розгалужених алгоритмів.

 Контрольні питання 

1. Опишіть синтаксис операторів організації розгалужених алгоритмів мови Pascal.
2. Опишіть особливості використання процедур виведення Read та ReadLn.
3. Опишіть особливості використання процедур введення Write та WriteLn.
4. Опишіть правила виклику процедури та функції.
5. Опишіть умови та випадки автоматичного перетворення типів даних.
6. Опишіть засоби явного перетворення типів даних.
7. Коротко опишіть процедури елементарного перетворення інформації: ціле↔порядкове, ціле↔дійсне, рядок↔символ.

 Індивідуальне завдання 

1. Наберіть в редакторі задану програму (Додаток 3). Виправте помилки та змініть її у такий спосіб, щоб вона виводила дані, які зазначено для Вашого варіанта.

Оцінюється якість організації інтерфейсу користувача: результат виводу повинен бути наочним та легко сприйматися користувачем.

№	Кількість змінних цілого, дійсного та символьного типів (в шаблоні – 2)	Розмір поля виводу (в шаблоні – 15)	Кількість знаків після десятинної точки (в шаблоні – 3)
X	3	16	4

2. Обчислити значення функції згідно з варіантом в точці. Вхідні дані – дійсне число, що є значенням аргументу; вихідні – дійсне число, що є значенням функції в точці.

Оцінюється якість організації інтерфейсу користувача, доцільність обраних типів даних та вміння використовувати шаблони програмування.

№	Функція
X	$f = \begin{cases} x, \text{якщо } x \geq 0; \\ -x, \text{в іншому випадку} \end{cases}$

3. Створити програму, що виводить похідну функції згідно з варіантом в зазначеній точці із визначеним шагом. Вхідні дані – два дійсні числа: перше – аргумент функції, друге – шаг аргументу функції; вихідні дані – дійсне число, що є значенням похідної у визначеній точці.

Оцінюється якість організації інтерфейсу користувача та доцільність обраних типів даних. Похідна функції в точці обчислюється за формулою $f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$.

№	Обчислення
X	$f(x) = \cos^2 x - \cos x^2$

4. Створити програму для введення та виведення змінної перерахункового типу даних. Смисл типу даних вказано за варіантом. Вхідні дані – номер константи перерахункового типу даних; вихідні дані – рядкове значення, що відповідає введеній константі.

Оцінюється якість організації інтерфейсу користувача, доцільність обраних типів даних та вміння використовувати шаблони програмування.

№	Смисл типу даних
X	Знак зодіаку

₪ Лабораторна робота № 4

- **Тема:** Цикли та робота з рядковим типом даних.
- **Мета:** здобуття навичок з обробки числових та логічних виразів за допомогою мови Pascal; вивчення шаблонів обробки послідовностей даних та роботи з рядками; організація циклічних алгоритмів.

Контрольні питання

1. Надайте визначення термінам «операція» та «операнд».
2. Для кожного базового типу даних опишіть допустимі операції.
3. Надайте пріоритет операцій мови Pascal.
4. Надайте змішані операції з описом типів операндів та результату.
5. Опишіть синтаксис та особливості оператора for..to|downto..do.
6. Опишіть синтаксис та особливості оператора while..do.
7. Опишіть синтаксис та особливості оператора repeat..until.
8. Опишіть функції роботи з рядками:
   I. Pos
   II. Delete
   III. Concat
   IV. Copy
   V. Insert
   VI. Length

Індивідуальне завдання

1. Підрахувати, скільки чисел в діапазоні від 0 до 999 999 задовольняють вимогу, зазначену за варіантом. Вхідні дані – немає; вихідні дані – число, що дорівнює кількості чисел, що задовольняють вимогу.

Існують два шляхи обчислень. Це завдання передбачає повний (або частковий) перебір чисел заданого діапазону. Але, якщо студент додатково вирішить завдання й іншим шляхом, це надасть йому додаткові бали за самостійну роботу. Оцінюється як обраний алгоритм та його реалізація, так і доцільність обраних даних та організація інтерфейсу користувача.

№	Умова	Приклад
X	Сума перших трьох цифр дорівнює сумі останніх трьох	234513

2. Створити програму, що виводить інтеграл функції згідно з варіантом в зазначених межах із визначеним шагом. Вхідні дані – три дійсні числа: перше та

друге – межі обчислення інтегралу, третє – шаг аргументу функції; вихідні дані – дійсне число, що є значенням інтеграла функції у визначених межах.

Оцінюється якість реалізації алгоритму, інтерфейс користувача та доцільність обраних типів даних. Інтеграл функції в межах обчислюється як сума за формулою $\int_{x_0}^{x_1} f(x)dx = \sum_{x=x_0}^{x_1} f(x)\Delta x$.

№	Функція
X	$f(x) = \sin x - \cos 2x$

3. Створити програму, що виводить перші N членів числової послідовності (послідовність зазначена за варіантом) та суму цих членів послідовності. Вхідні дані: три цілі числа: перше – кількість членів послідовності, друге та третє – нульовий та перший члени послідовності; вихідні дані – числа-члени послідовності та число, що є сумою членів послідовності.

Оцінюється якість реалізації алгоритму, інтерфейс користувача та доцільність обраних типів даних.

№	Послідовність
X	$a_n = 2 \cdot a_{n-1} - 3 \cdot a_{n-2}$

4. Задана послідовність, що містить від 2 до 20 слів, в кожному із них від 1 до 10 символів, між сусідніми словами не менше одного символу відступу (пробіл або табуляція), за останнім словом – крапка. Вивести всі слова послідовності, що задовольняють умови згідно з варіантом. Вхідні дані – послідовність символів; вихідні – послідовності символів, що представляють слова вхідної послідовності та задовольняють вимоги.

Оцінюється обраний алгоритм вирішення завдання, інтерфейс користувача та доцільність обраних типів даних. Реалізація алгоритму не повинна містити звернення до рядка як до масиву – тільки стандартні функції роботи з рядками.

№	Умови
X	Усі слова, що містять перший символ.

₪ Лабораторна робота № 5

- **Тема:** Обробка масивів даних. Типові алгоритми пошуку та впорядкування. Організація підпрограм. Рекурсія та ітерація. Механізм реалізації рекурсії.
- **Мета:** здобуття навичок з обробки масивів за допомогою мови Pascal; вивчення алгоритмів пошуку та впорядкування; здобуття навичок зі структуризації програми та організації підпрограм; знайомство з механізмом рекурсії.

 **Контрольні питання** 

1. Опишіть синтаксис визначення масивів даних.
2. Опишіть засоби ініціалізації масивів.
3. Надайте алгоритми пошуку елемента в масиві даних. Назвіть умови використання, позитивні та негативні сторони алгоритму (*лінійний* та *бінарний* пошуки).
4. Надайте алгоритми пошуку підпослідовності в послідовності даних. Вкажіть умови використання, позитивні та негативні сторони алгоритму *(прямий* пошук, алгоритми *Боуера–Мура* та *Кнута–Морріса–Пратта).*
5. Надайте алгоритми упорядкування даних. Назвіть умови використання, позитивні та негативні сторони алгоритму (алгоритми *прямого вибору*, *бінарної вставки*, *обміну (бульбашка)*, *шейкерна*, *Шелла*, *пірамідальна).*
6. Опишіть правила визначення підпрограм. Надайте синтаксис визначення процедур та функцій, умови використання процедур та функцій.
7. Надайте визначення термінів «формальний параметр підпрограми» та «фактичний параметр підпрограми».
8. Опишіть типи формальних параметрів підпрограм (параметр-значення, параметр-змінна, параметр-константа). Як передаються фактичні параметри різних типів. Доцільність використання кожного типу параметра.
9. Надайте визначення термінів «локальний контекст» та «глобальний контекст». Наведіть приклад.
10. Надайте визначення терміна «рекурсія». Наведіть приклад рекурсивної підпрограми. Мета та доцільність використання рекурсії.
11. Опишіть типи рекурсії.

 **Індивідуальне завдання** 

1. Написати програму, що виводить мінімум та максимум функції на заданому відрізку. Вхідні дані – два дійсні числа: нижнє та верхнє значення аргументу функції; вихідні дані – чотири дійсні числа, що є мінімумом та максимумом функції на заданому відрізку, обчислених за допомогою ітераційного та рекурсивного алгоритмів.

Функція для пошуку екстремумів та ж сама, що і в завданні 2 лабораторної роботи № 4. В програмі потрібно виділити окремі функції для ітераційного та рекурсивного рішень. Оцінюється якість реалізації алгоритму, організація інтерфейсу користувача, доцільність використаних типів даних.

2. Написати програму, що перетворює квадратну дійсну матрицю згідно з варіантом. Вхідні дані – розмірність матриці та елементи матриці; вихідні дані – перетворена матриця.

Розмірність матриці обов'язково має вводитися користувачем і повинна бути не менш ніж 50. Програма повинна надавати користувачу можливість вибору створення масиву даних: ручне введення або заповнення довільними значеннями (генерація). Оцінюється як обраний алгоритм та його реалізація, так і доцільність обраних даних та організація інтерфейсу користувача.

№	Умови	
X	Віддзеркалення матриці відносно основної діагоналі	$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \Rightarrow \begin{vmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{vmatrix}$

3. На основі отриманого шаблона реалізувати алгоритми пошуку: лінійний та бінарний пошук у цілочисельному масиві та прямий пошук підрядка.

Дотримуючись шаблона, програма повинна правильно виводити кількість проведених порівнянь. При захисті лабораторної роботи студент повинен знати і вміти розповісти кожен з реалізованих алгоритмів. Оцінюється реалізація та знання алгоритмів пошуку. Допустимою є й інша реалізація алгоритмів Боуера–Мура та Кнута–Морріса–Пратта, що оцінюється додатково.

4. На основі отриманого шаблону реалізувати алгоритми упорядкування: прямого вибору, бінарної вставки, обміну (бульбашка), шейкерна.

Дотримуючись шаблона, програма повинна правильно виводити кількість проведених порівнянь. При захисті лабораторної роботи студент повинен знати і вміти розповісти кожен з реалізованих алгоритмів. Оцінюється реалізація та знання алгоритмів упорядкування. Допустимою є й інша реалізація алгоритмів упорядкування Шелла (пірамідальний та швидкий рекурсивний), що оцінюється додатково.

₪ Лабораторна робота № 6

- **Тема:** Тип даних запис. Перерахунковий та діапазонний типи даних. Файловий тип даних. Етапи роботи з файлом. Операції над файлом.
- **Мета:** здобуття навичок з визначення типів даних користувача; тренування з читабельності програм; здобуття навичок з обробки файлів.

Контрольні питання

1. Опишіть синтаксис діапазонного типу даних. Мета та доцільність використання діапазонного типу.
2. Опишіть синтаксис перерахункового типу даних. Мета та доцільність використання перерахункового типу.
3. Опишіть операції для роботи з перерахунковим типом даних.
4. Введення та виведення змінних перерахункових типів даних.
5. Опишіть синтаксис типу даних запис. Представлення змінних типу запис в пам’яті.
6. Введення та виведення змінних типу запис.
7. Опишіть синтаксис файлового типу даних. Види файлів в Turbo Pascal 7.0 та доцільність вибору того чи іншого виду файлового типу.
8. Опишіть етапи роботи з файлом.
9. Опишіть функції для роботи з текстовими файлами.
10. Опишіть функції для роботи з типізованими файлами.
11. Опишіть функції для роботи з нетипізованими файлами.
12. Опишіть форму зберігання інформації у файлах.

Індивідуальне завдання

1. Написати програму для пошуку файлів, що задовольняють вимоги щодо варіанта в заданому каталозі. Вхідні дані – рядок, що задає каталог пошуку; вихідні дані – текстовий файл output.txt, що містить повні імена файлів, які задовольняють завдання.

Оцінюється оптимальність реалізації алгоритму. Доцільно використовувати рекурсивний алгоритм обходу дерева каталогів. Додаткова реалізація ітераційного алгоритму оцінюється окремо.

№	Об’єкти
X	Усі приховані файли та каталоги.

2. Написати програму, яка виводить в текстовий файл всі члени числової послідовності, що відрізняються від попереднього більш ніж на задане число. Вхідні дані: перший член послідовності та число, що задає мінімальну

різницю між членами послідовності. Вихідні дані: текстовий файл output.dat, в кожному рядку якого зазначено номер члена послідовності та його значення.

Оцінюється вміння роботи з текстовими файлами, організація інтерфейсу користувача та доцільність обраних типів даних та змінних. Програма повинна відповідати угорській нотації та принципу модульності. Програма повинна перевіряти, щоб при заданому першому члені послідовність збігалася.

№	Послідовність
X	$a_n = \ln(\lvert a_{n-1} \rvert)$

3. Написати програму, що дозволяє вносити та проглядати дані згідно з варіантом. Під час реалізації обов'язково використовувати тип даних запис, діапазонні та перерахункові типи. Вхідні дані: режим роботи (внесення або перегляд) та, якщо внесення, необхідні дані щодо запису. Вихідні дані: змінений файл, при внесенні та таблиця з усіма внесеними даними на екрані або у текстовому файлі (за вибором користувача), при перегляді даних.

Програма повинна зберігати інформацію в типізованому файлі. Оцінюється розроблений тип даних, реалізація алгоритмів обробки даних та організація інтерфейсу користувача. Реалізація додаткових функцій обробки (видалення, редагування та пошук записів) оцінюється додатково.

№	Набір даних	Обов'язкові атрибути
X	Каталог меблів	Тип, назва, виробник, матеріал.

4. Заданий файл, компоненти котрого – цілі шестизначні числа. Видалити з файлу всі числа, що не задовольняють вимогу, зазначену за варіантом. Вхідні дані – ім'я файлу з числами; вихідні дані – змінений файл та час роботи програми.

Вимога щодо чисел – та ж сама, що у завданні 1 лабораторної роботи № 4. Програма повинна працювати з нетипізованим файлом, читати та писати дані блоками заданого розміру. Виконання аналізу щодо оптимального розміру блоку надає додаткові бали. Також програма не повинна створювати вхідний файл – для створення вхідного файлу та перегляду вихідного потрібно створити додаткову програму. Також не допускається використання допоміжних файлів – програма повинна працювати лише з одним файлом.

₪ Лабораторна робота № 7

- **Тема:** Вказівники та адреса. Операції з динамічною областю пам'яті. Організація модулів.
- **Мета:** здобуття навичок з обробки даних за допомогою динамічної пам'яті; ознайомлення з перевагами повторного використання коду.

Контрольні питання

1. Опишіть синтаксис вказівників. Мета та доцільність використання вказівникового типу даних.
2. Як формується вказівник в Turbo Pascal? Надайте визначення понять сегмент і зсув та процедури розкладення та побудови вказівників.
3. Опишіть особливості нетипізованих вказівників.
4. Поясніть зміст стандартних змінних HeapOrg, HeapEnd, HeapPtr.
5. Опишіть роботу процедур New(…) та Dispose(…).
6. Опишіть роботу процедур Mark(…) та Release(…).
7. Опишіть роботу процедур Getmem(…) та Freemem(…).

Індивідуальне завдання

1. Написати програму, що розташовує в динамічній області пам'яті одновимірний масив дійсних чисел розміром до 75 000 елементів та обчислює значення згідно з варіантом. Вхідні дані – розмір масиву та засіб ініціалізації; вихідні дані – число, що відповідає завданню.

Програма повинна реалізовувати три засоби ініціалізації масиву: заповнення довільними значеннями (генерація), введення з клавіатури та введення з текстового файлу. Оцінюється оптимальність реалізації алгоритму, правильність роботи з вказівниками та інтерфейс користувача.

№	Завдання
X	Кількість негативних елементів

2. Написати програму для перемноження двох дійсних квадратних матриць, розмір яких може бути від 1×1 до 150×150. Вхідні дані – розмір матриць та засіб ініціалізації; вихідні дані – текстовий файл output.txt, що містить результат перемноження матриць.

Матриці потрібно розташовувати в динамічній області пам'яті. Програма повинна реалізовувати три засоби ініціалізації матриць: заповнення довільними значеннями (генерація), введення з клавіатури та введення

з текстового файлу. Оцінюється оптимальність реалізації алгоритму, правильність роботи з вказівниками та інтерфейс користувача.

3. Написати програму, що розташовує в динамічній області пам’яті одновимірний масив чисел розміром до 75 000 елементів за допомогою нетипізованого вказівника та обчислює значення згідно з варіантом. Елементами масиву є числа довільного числового типу (short, integer, single, real, double). Вхідні дані – розмір масиву, тип елемента та засіб ініціалізації; вихідні дані – число, що відповідає завданню.

Програма повинна реалізовувати три засоби ініціалізації масиву: заповнення довільними значеннями (генерація), введення з клавіатури та введення з текстового файлу. Оцінюється оптимальність реалізації алгоритму, правильність роботи з вказівниками та інтерфейс користувача. Варіанти завдання ті ж самі, що у завданні.

4. Написати програму, що за допомогою нетипізованого вказівника дозволяє розташувати в пам’яті текст довільного розміру (до 64 Кб) та вивести всі слова тексту, що задовольняють умови згідно з варіантом. Вхідні дані – ім’я файлу з текстом; вихідні дані – текстовий файл, що містить тільки ті слова, що задовольняють вимоги варіанта.

Вимога до варіантів та ж сама, що у завданні 4 лабораторної роботи № 4. Оцінюється оптимальність реалізації алгоритму та правильність роботи з вказівниками.

Лабораторна робота № 8

- **Тема:** Формування списків. Операції над списками. Організація модулів. Вступ до об’єктно-орієнтованого програмування.
- **Мета:** здобуття навичок з обробки списків; ознайомлення з абстрактними типами даних; ознайомлення з перевагами повторного використання коду; ознайомлення з класами та об’єктами.

 **Контрольні питання** 

1. Опишіть засоби реалізації односпрямованого списку та принципи роботи з ним.
2. Опишіть засоби реалізації двонаправленого списку та принципи роботи з ним.
3. Дайте визначення понять «Стек», «Черга», «Дерево», «Асоціативний масив (Словник)».
4. Опишіть структуру модуля Turbo Pascal. Надайте опис кожного блоку модуля.
5. Надайте визначення понять «Клас», «Об’єкт».
6. Опишіть засоби реалізації об’єктів в Turbo Pascal.
7. Надайте визначення понять «Член класу», «Властивість класу», «Метод класу».
8. Опишіть модифікатори доступу до членів класу.
9. Надайте визначення понять «Абстракція», «Інкапсуляція», «Спадкування», «Поліморфізм».

 **Індивідуальне завдання** 

1. Написати програму, що дозволяє вносити та проглядати дані згідно з варіантом. Під час реалізації обов’язково використовувати тип даних запис, діапазонні та перерахункові типи. Вхідні дані: режим роботи (внесення або перегляд) та, якщо внесення, необхідні дані щодо запису. Вихідні дані: змінений файл, якщо внесення, та таблиця з усіма внесеними даними на екрані або у текстовому файлі (за вибором користувача) при перегляді даних.

   
   Програма повинна зберігати інформацію в типізованому файлі. В процесі роботи програми дані повинні повністю завантажуватись в пам’ять програми та зберігатися у вигляді односпрямованого списку. Оцінюється розроблений тип даних списку, реалізація алгоритмів обробки списку та організація інтерфейсу користувача. Реалізація інших функцій обробки (видалення, редагування та пошук записів) оцінюється додатково. Індивідуальне завдання те ж саме, що у завданні 3 лабораторної роботи 6.
2. Написати модуль, що реалізує тип даних «Список» та функції обробки цього списку.

Модуль повинен містити тип даних «Список» та базові функції: створення, видалення, пошук елемента за значенням, додавання елемента, виведення на екран. Модуль повинен супроводжуватись програмою, що дозволяє його тестувати Оцінюється вміння роботи з динамічною пам'яттю та повнота й інтерфейс користувача супровідної програми.

№	Тип списку	Тип елемента
X	Односпрямований	Byte

3. Написати модуль, що реалізує клас «Список».

Модуль повинен містити тип даних «Список», що є класом та інкапсулює структуру списку та базові функції: створення, видалення, пошук елемента за значенням, додавання елемента, виведення на екран. Модуль повинен супроводжуватись програмою, що дозволяє його тестувати Оцінюється вміння роботи з динамічною пам'яттю, організація класу та повнота й інтерфейс користувача супровідної програми. Тип списку та тип елемента списку – ті ж самі, що у завданні 2.

4. Написати модуль, що реалізує класи «Черга» та «Стек».

Модуль повинен містити класи «Черга» та «Стек», що є спадкоємцями абстрактного класу «Колекція». Клас «Колекція» інкапсулює структуру «Список» та базові функції обробки списку. Також клас «Колекція» оголошує наявність двох методів «Put» та «Pop», що реалізуються в класах «Черга» та «Стек» окремо. Модуль повинен супроводжуватись програмою, що дозволяє його тестувати. Оцінюється вміння роботи з динамічною пам'яттю, організація ієрархії класів та повнота й інтерфейс користувача супровідної програми. Тип списку та тип елемента списку – ті ж самі, що у завданні 2.

Розділ 2. Довідкова інформація

Базові терміни

Алгоритм (рос. *алгоритм;* англ. *algorithm*) – попередньо задана послідовність чітко визначених правил або команд для одержання рішення задачі за кінцеве число кроків.

Введення (рос. *ввод;* англ. *input*) – процес введення інформації.

Виведення (рос. *вывод;* англ. *output*) – процес передавання інформації на зовнішній пристрій.

Вираз (рос. *выражение;* англ. *expression*) – мовна конструкція для обчислення значення за допомогою одного або декількох операндів.

Вказівник (рос. *указатель;* англ. *pointer*) – тип даних, змінна котрого вказує місцезнаходження першого байта (слова) в пам'яті, з якого починається елемент, що адресується.

Дані (рос. *данные;* англ. *data*) – інформація, підготовлена для певних цілей у певному форматі.

Змінна (рос. *переменная;* англ. *variable*) – ідентифікатор, який використовується для позначення деякої величини, що зберігається, яка може змінюватися під час виконання програми.

Ідентифікатор (рос. *идентификатор;* англ. *identifier*) – лексична одиниця, яка використовується як ім'я для елементів мови; ім'я, яке присвоюється об'єкту та являє собою послідовність латинських букв та цифр, яка починається з букви.

Ініціалізація (рос. *инициализация;* англ. *initialization*) – установка програмних змінних в нуль або задання їм інших початкових значень перед виконанням основного алгоритму.

Інформатика (рос. *информатика;* англ. *computer science, informatics*) – науковий напрям, який займається вивченням законів, методів та способів накопичення, обробки і передачі інформації за допомогою ЕОМ та інших технічних засобів; група дисциплін, яка займається різними аспектами використання та розробки ЕОМ: прикладна математика, програмування, програмне забезпечення, штучний інтелект, архітектура ЕОМ, обчислювальні мережі тощо.

Ітерація (рос. *итерация;* англ. *iteration*) – процес обчислень, заснований на повторені послідовності операцій, при якому на кожному кроці повторення використовується результат попереднього кроку.

Клас (рос. *класс;* англ. *class*) – абстрактний тип даних, що об'єднує інформацію із засобами її обробки.

Компонувальник (рос. *компоновщик;* англ. *linker*) – завантажувач, який виконує при завантаженні компонування єдиної програми з програм, що були трансльовані незалежно.

Константа (рос. *константа;* англ. *constant*) – вираз, що не змінюється ні за яких умов.

Масив даних (рос. *массив данных;* англ. *data array*) – упорядкований набір однотипних елементів, число яких фіксоване.

Мова програмування (рос. *программирования;* англ. *programming language*) – система позначень, потрібна для точного опису програм та алгоритмів для ЕОМ. Є штучною мовою, в якій синтаксис та семантика строго визначені.

Налагоджувальник (рос. *отладчик;* англ. *debugger*) – програма, або частина середовища розробки для пошуку логічних помилок у програмі.

Об'єкт (рос. *объект;* англ. *object*) – це (в об'єктно-орієнтованому програмуванні) змінна або екземпляр якогось класу.

Об'єктно-орієнтоване програмування (рос. *объектно-ориентированное программирование*; англ. *object-oriented programming*) – парадигма програмування, що передбачає використання об'єктно-орієнтованого підходу щодо етапів розробки програми: об'єктно-орієнтована мова програмування, об'єктно-орієнтоване середовище розробки, об'єктно-орієнтовані засоби проектування.

Обробка даних (рос. *обработка данных;* англ. *data processing*) – послідовна переробка пакетів інформації, при якій відносна кількість груп даних піддається обробці за тими самими процедурами.

Операнд (рос. *операнд;* англ. *operand*) – якісна величина, над якою проводиться математична або логічна операція.

Оператор (рос. *оператор;* англ. *statement*) – блок, який використовується при створенні програм мовою високого рівня: програма є послідовністю речень-операторів.

Операція (рос. *операция;* англ. *operation*) – функція, яка визначена на деякій підмножині S в степені m множини S, специфічної для цієї функції.

Парадигма програмування (рос. *парадигма программирования;* англ. *programming paradigm*) – схема розробки програмного забезпечення, що містить модель постановки проблем та їхнє вирішення, засоби та технології, що використовуються під час розробки.

Підпрограма (рос. *подпрограмма;* англ. *subroutine*) – частина програми, яка виконується «поза чергою», тобто керування передається підпрограмі, а після її закінчення повертається на ту команду головної програми, яка йде за оператором виклику.

Повторне використання коду (рос. *повторное использование кода;* англ. *code reuse*) – технологія, згідно з якою окремі підзадачі головної задачі оформлюються у вигляді окремих модулів та можуть бути використані при вирішенні східних підзадач без змін.

Пошук (рос. *поиск;* англ. *searching*) – процес відшукання інформації у таблиці або у файлі шляхом перегляду спеціального поля кожного запису, яке називається ключем.

Присвоєння (рос. *присвоение;* англ. *assignment*) – основний оператор всіх мов програмування, окрім декларативних мов, за допомогою яких змінній присвоюється нове значення.

Програма (рос. *программа;* англ. *program*) – набір операторів, який може бути поданий як одне ціле в деякій обчислювальній системі та використовується для керування цією системою. Реалізація методу обчислень мовою високого рівня.

Програмування (рос. *программирование;* англ. *programming*) – всі технічні операції, необхідні для створення програми, включно з аналізом вимог та всіма стадіями розробки й реалізації.

Процедура (рос. *процедура;* англ. *procedure*) – частина програми, яка виконує деяку чітко визначену операцію над даними, може бути викликана з будь-якого місця програми, й, залежно від параметрів, видавати різні результати. Щодо мов програмування, процедурою називається підпрограма, що використовується як окремий оператор.

Рекурсія (рос. *рекурсия;* англ. *recursion*) – процес визначення функції, мовної конструкції або розв'язку задачі через самих себе, що приводить до появи рекурсивної функції, рекурсивної задачі або рекурсивної мовної конструкції.

Середовище розробки (рос. *среда разработки;* англ. *developer's environment*) – обчислювальна система, що призначення для автоматизації задач, пов'язаних з розробкою програм.

Сортування (рос. *сортировка;* англ. *sorting*) – процес переупорядкування інформації за певним порядком (зростання чи спадання) ключів сортування.

Список (рос. *список;* англ. *list*) – деяка обмежена послідовність пунктів ($x_1, x_2, \ldots, x_n$), де $n \geq 0$. Щодо абстрактних типів даних список – це колекція елементів, що пов'язані однин з одним у певному порядку.

Структурне програмування (рос. *структурное программирование;* англ. *structured programming*) – парадигма програмування, що передбачає утворення зрозумілих, локально простих та зручних для читання програм, характерними особливостями яких є модульність, використання уніфікованих структур слідування, вибору та повторення, відмову від неструктурованих передач керування, органічне використання глобальних змінних.

Тип даних (рос. *тип данных;* англ. *data type*) – домовленість щодо того, як буде інтерпретуватись та оброблятись послідовність одиниць пам'яті для представлення інформації певної природи.

Точка переривання (рос. *точка прерывания;* англ. *breakpoint*) – місце в програмі, де може бути перерване її виконання.

Транслятор (рос. *транслятор;* англ. *translator*) – програма або частина середовища розробки для перетворення програми, написаної однією мовою програмування, в програму, написану іншою мовою. Наприклад, з програми мовою Pascal на машинний код.

Функція (рос. *функция;* англ. *function*) – відбиття (перетворення) однієї множини X на (в) іншу множину Y. Щодо мов програмування, функцією називається підпрограма, що може використовуватися у виразах та повертає одне значення.

Цикл (рос. *цикл;* англ. *loop, cycle*) – набір операцій, який регулярно повторюється в одній і тій же послідовності.

Шаблон програмування (рос. *шаблон программирования;* англ. *programming pattern*) – конструкція, або фрагмент коду, що багаторазово використовується для вирішення загальної проблеми програмування.

Концепція типів даних

Уся інформація у пам'яті комп'ютера зберігається у вигляді послідовності чарунок, що мають значення або 0, або 1 **(біти).** *Біти є мінімальною одиницею пам'яті, яку можна обробляти*. Група з 8 бітів називається **байтом**. *Байт є мінімальною одиницею пам'яті, яку можна зберігати*. Для зберігання та обробки інформації різної природи було введено поняття типу даних.

Тип даних – це домовленість щодо того, як буде *інтерпретуватись* послідовність бітів для представлення інформації *певної природи*. Тип даних описує *кількість бітів,* потрібних для представлення інформації, як ці біти потрібно *інтерпретувати*, *область допустимих значень* – всі значення, що можуть бути представлені цим типом, та *операції,* що допустимо виконувати над інформацією.

Типи даних прийнято розподіляти на **базові** (або **примітивні**) типи даних та типи даних **користувача.** Примітивні типи даних представляють найбільш універсальні різновиди інформації – числа, символи тощо.

Далі надано опис типів даних, що використовуються в Turbo Pascal. В інших мовах програмування можуть використовуватися інші типи, або може бути змінено параметри цих типів даних, проте поняття типу даних є універсальним, і в більшості мов програмування концепція типів подібна наданій.

⇒ **Базові типи даних**

- **Порядкова інформація**
  - **Цілі числа**

Для представлення цілих чисел використовується звичайний перелік значень, що можуть бути представлені бінарними числами певної довжини. Так, наприклад, 4-ма бітами може бути представлено 16 різних значень. Проте, якщо перший лівий біт буде означати знак числа (0 – це +, 1 – це –), то діапазон можливих значень зміщується відносно 0. Нижче наведено приклад типу даних, що використовує 4 біти для представлення цілого числа.

2	10	10 (+/–)	16	2	10	10 (+/–)	16
0000	0	0	0	1000	8	–8	8
0001	1	1	1	1001	9	–7	9
0010	2	2	2	1010	10	–6	A
0011	3	3	3	1011	11	–5	B
0100	4	4	4	1100	12	–4	C
0101	5	5	5	1101	13	–3	D
0110	6	6	6	1110	14	–2	E
0111	7	7	7	1111	15	–1	F

Перший стовбець – бінарне представлення числа, другий – десятинне представлення числа без знаку, третій – десятинне представлення числа зі знаком,

четвертий – шістнадцяткове представлення числа. Нижче наведено таблицю з характеристикою всіх цілих типів, що використовуються в Turbo Pascal.

Назва типу	Кількість байтів	Мінімальне значення	Максимальне значення	Розташування в пам'яті
byte	1	0	+255	8 біт – значення
shortint	1	–128	+127	1 біт – знак, 7 біт – значення
word	2	0	+65 535	16 біт – значення
integer	2	–32768	+32 767	1 біт – знак, 15 біт – значення
longint	4	–2147483648	+2147483647	32 біт – значення

– **Логічна інформація**

У Turbo Pascal логічна інформація представлена одним типом – **boolean.** Цей тип в Turbo Pascal є попередньо визначеним перерахунковим типом, який має два можливі значення – true та false. Займає 1 байт. Порядкові номери констант – false – 0; true – 1.

– **Символи**

У Turbo Pascal символьна інформація представлена одним типом – **char.** Цей тип ставить відповідно до кожного числа з діапазону типу **byte** якийсь символ. Ця відповідність має назву «Таблиця ASCII символів» (англ. American Standard Code for Information Interchange – американський стандартний код для обміну інформацією). ASCII є 7-бітним кодуванням десятинних цифр, латинського й національного алфавітів, знаків пунктуації та керуючих символів. В ком-п'ютерах звичайно використовують 8-бітні розширення ASCII. Нижче надана 8-бітна таблиця ASCII символів, локалізована для кирилиці (російська та українська).

	***0**	***1**	***2**	***3**	***4**	***5**	***6**	***7**	***8**	***9**	***A**	***B**	***C**	***D**	***E**	***F**
0*	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1*	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2*		!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3*	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4*	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5*	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6*	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7*	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8*	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
9*	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
A*	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
B*	▒	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C*	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
D*	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	▀	▄	▌	▐	■
E*	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F*	Ё	ё	Є	є	Ї	ї	Ў	ў	°	∙	·	√	№	¤	■	

Оскільки ASCII спочатку призначався для обміну інформацією (по телетайпу), в ньому, окрім інформаційних символів, використовуються символи-команди для керування зв'язком. Це звичайний набір спеціальних сигналів, що використовувався і в інших докомп'ютерних засобах обміну повідомленнями (азбука Морзе, семафорна азбука), доповнений з урахуванням специфіки пристрою.

NUL – *Null,* порожній. Завжди ігнорувався. На перфострічках 1 відмічалася дірочкою, 0 – відсутністю дірочки. Отже, порожні частини перфострічки до початку та після закінчення повідомлення складалися з таких символів. Зараз використовується у багатьох мовах програмування як кінець рядка.

SOH – *Start Of Heading,* початок заголовка.

STX – *Start of Text,* початок тексту.

ETX – *End of Text,* кінець тексту. Зараз використовується як символ Ctrl–C, для зупинення роботи чогось.

EOT – *End of Transmission,* кінець передавання. В системі UNIX Ctrl–D, яка має той самий код, означає кінець файлу при введенні з клавіатури.

ENQ – *Enquire*. Прошу підтвердження.

ACK – *Acknowledgement*. Підтверджую.

BEL – *Bell,* дзвоник, звуковий сигнал. Зараз теж використовується.

BS – *Backspace,* повернення на один символ. Зараз стирає попередній символ.

TAB (**HT**) – *Tabulation* (*Horizontal Tabulation*). Горизонтальна табуляція.

LF – *Line Feed,* переведення рядка. Зараз в кінці кожного рядка текстового файлу ставиться або цей символ, або CR, або разом (CR, після LF).

VT – *Vertical Tabulation,* вертикальна табуляція.

FF – *Form Feed,* нова сторінка.

CR – *Carriage Return,* вертання каретки. У багатьох мовах програмування цей символ можна використовувати для повернення на початок рядка без переведення рядка.

SO – *Shift Out,* зміна кольору стрічки (використовувався для двокольорових стрічок; колір зазвичай змінювався на червоний).

SI – *Shift In*, навпаки Shift Out.

DLE – *Data Link Escape,* наступні символи мають спеціальний зміст.

DC1–4 – *Device Control 1–4,* 1–4-й символ керування пристроєм (1 – увімкнути пристрій читання перфострічки, 2 – увімкнути перфоратор, 3 – вимкнути пристрій читання перфострічки, 4 – вимкнути перфоратор).

NAK – *Negative Acknowledgment*, не підтверджую.

SYN – *Synchronization*. Цей символ передавався, коли для синхронізації було необхідно щось передати.

ETB – *End of Text Block*, кінець текстового блоку. Іноді текст з технічних причин розбивався на блоки.

CAN – *Cancel*, відміна (того, що було передано раніше).

EM – *End of Medium*, закінчилася перфострічка.

SUB – *Substitute*, підставити. Наступний символ – іншого кольору або з додаткового набору символів. Зараз цей код як символ Ctrl–Z використовується як кінець файлу при введенні з клавіатури в системах DOS та Windows. Ця функція немає ніякого зв'язку з символом SUB.

ESC – *Escape*. Наступні символи – дещо спеціальне.

FS, GS, RS, US – *File, Group, Record, Unit Separator*, символ, що розділяє файли, групи, записи та модулі. Тобто підтримувалося 4 рівні структуризації даних: повідомлення могло складатися з файлів, файли з груп, групи з записів, записи з модулів.

DEL – *Delete*, видалити останній символ. Символом DEL, що складається в бінарному коді зі всіх одиниць (у 7-бітному кодуванні), можна було забити будь-який символ. Пристрої та програми ігнорували DEL, як і NUL.

- **Дійсні числа**

Для представлення дійсних чисел використовується така схема: будь-яке число записується у вигляді з рухомою крапкою: $123 = 1{,}23 \cdot 10^2$; $23{,}456 = 2{,}3456 \cdot 10^1$; $9{,}076 = 9{,}076 \cdot 10^0$; $0{,}0015 = 1{,}5 \cdot 10^{-3}$. Тобто у вигляді $\mathbf{X \cdot 10^Y}$, де X – число більше за 0 та менше за 10 – це є мантисою дійсного числа, а Y – ціле число – є порядком дійсного числа. Основними характеристиками дійсних типів даних є: діапазон можливих значень – визначає головним чином діапазон порядку та точність представлення даних – скільки цифр числа може бути збережено. Нижче наведено таблицю з характеристикою всіх дійсних типів, що використовуються в Turbo Pascal. В таблиці розташування в пам'яті наведено за наступною формулою: кількість біт – поле, де поле є: S – знак числа; E – показник порядку числа; F – мантиса числа; I – ціла частина числа.

Назва типу	Кількість байтів	Діапазон	Точність	Розташування в пам'яті
real	6	$2.9 \cdot 10^{-39}..1.7 \cdot 10^{+38}$	11..12	
single	4	$1.5 \cdot 10^{-45}..3.4 \cdot 10^{+38}$	7..8	1S-8E-23F
double	8	$5.0 \cdot 10^{-324}..1.7 \cdot 10^{+308}$	15..16	1S-11E-52F
extended	10	$3.4 \cdot 10^{-4932}..1.1 \cdot 10^{+4932}$	19..20	1S-15E-1I-63F
comp	8	$-2^{63}+1..2^{63}-1$	19..20	

Тип real спеціально розроблено для тих випадків, коли процесор працює без співпроцесора, який обробляє операції з рухомою крапкою (селектор {$N–}). В іншому випадку виправдано та доцільно використовувати типи single та double (селектор {$N+}). Використання типів extended та comp можливе тільки при

ввімкнутому співпроцесорі (селектор {$N+}). Тип даних comp є представленням цілого числа в дійсній формі.

- **Рядковий тип**

Рядки у мовах програмування завжди представляються як масив символів. Існує декілька засобів організації рядків:

– фіксованого максимального розміру з динамічною довжиною визначається як масив символів з межами 0..МАКС, де МАКС – максимальна можлива кількість символів в рядку. МАКС задається користувачем, в іншому випадку встановлюється за замовчуванням (в Turbo Pascal – 255). Номер символу з індексом 0 інтерпретується як динамічна довжина рядка (наприклад, якщо символ з індексом 0 є «0», то в рядку 48 символів), інші символи складають саме рядок;

– динамічний рядок, що закінчується нулем під рядок виділяється певна кількість пам'яті, в яку записуються символи. Перший символ з номером 0 вважається кінцем рядка. Такі рядки обов'язково повинні закінчуватися символом з номером 0, інакше станеться вихід за межі символьного масиву й інформація, що не стосується рядка, буде інтерпретуватися як рядок.

⇒ **Типи даних користувача**

- **Перерахунковий**

Перерахункові типи даних – це основний інструмент створення коду програми, що легко читається та сприймається. Крім того, перерахункові типи даних – зручний інструмент для виключення багатьох логічних помилок у програмі. Цей тип даних належить також до порядкових типів даних.

Перерахунковий тип даних задається перерахунком тих значень, котрі він може отримувати. Кожне значення іменується деяким ідентифікатором та розташовується у списку в круглих дужках:

```
<ім'я_типу>=(<ідентифікатор1>,<ідентифікатор2>…,<ідентифікаторN>);
```

де (ідентифікатор1, ідентифікатор2, …, ідентифікаторN) – перелік можливих значень змінної цього типу – константи перерахункового типу. Наприклад, тип, що описує місяці, може бути упорядкований у такий спосіб:

```
TMonth=(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
```

Кожна константа перерахункового типу має свій номер, починаючи з 0, тобто ord(jan) = 0, а ord(dec) = 11. Максимальна кількість констант перерахункового типу – 65 536. Змінна цього типу займає 2 байти.

- **Діапазон**

Тип-діапазон – це засіб обмеження переліку можливих значень одного з порядкових типів (цілі, символи, перерахункові). Це може бути використано, коли чітко відомо, в якому діапазоні будуть змінюватися значення змінної, наприклад – межі масиву.

Діапазонний тип даних задається межами своїх значень в межах базового типу:

```
<ім'я_типу>=<мінімальне_значення>..<максимальне_значення>;
```

де мінімальне значення та максимальне значення – межі можливих значень змінної цього типу – константи одного з порядкових типів. Базовий тип даних визначається за константами мінімальне_значення та максимальне_значення. Наприклад, типи, що описують кількість днів в місяці, та місяці одного сезону (літа) можуть бути упорядковані у такий спосіб:

```
TDaysInMonth=1..31;
TSummerMonthes=jun..aug;
```

Змінна типу-діапазону займає стільки ж пам'яті, скільки й змінна базового для цього діапазону типу.

- **Структуровані типи даних**
  - **Масив**

Масив – це засіб упорядкування послідовності значень одного типу даних. Діапазонний тип даних подається у такий спосіб:

```
<ім'я типу>=array [<список індексних типів>] of <тип елементу>;
```

де список індексних типів – перераховані через кому один чи декілька типів індексів масиву, типом індексу масиву може бути тільки порядковий тип даних; тип елементу – тип даних кожного елемента послідовності.

Наприклад, масив з 10 цілих чисел подається як

```
arr: array [1..10] of integer;
```

Звернення до змінної як arr – звернення до масиву. Для звернення до конкретного елемента масиву виконується як arr[n], де n – індекс (номер) елемента масиву, до якого виконується звернення.

Типом елемента масиву може бути будь-який тип даних Turbo Pascal, включно з масивами. Записи

```
TArr=array [1..N] of array [1..M] of TElement;
```

та

```
TArr=array[1..N,1..M] of TElement;
```

з точки зору компілятора ідентичні.

У пам’яті масив займає потрібну кількість суміжних блоків розміром, достатнім для зберігання змінної типу елемента. Тобто, змінна

```
a: array [1..5] of double;
```

буде займати 40 байт (5 блоків по 8 байт):

Елементи масиву	a[1]	a[2]	a[3]	a[4]	a[5]
Індекси елементів	1	2	3	4	5

Якщо масив має декілька індексів, то в пам’яті масив знаходиться блоками, замінюючи правий індекс першим:

```
a: array [1..3,0..2] of double;
```

Елементи	a[1,0]	a[1,1]	a[1,2]	a[2,0]	a[2,1]	a[2,2]	a[3,0]	a[3,1]	a[3,2]
Індекси	1,0	1,1	1,2	2,0	2,1	2,2	3,0	3,1	3,2

Масив не може займати більш ніж 65 536 байт.

– **Запис**

Запис – це структура даних, що складається з фіксованої кількості компонентів, які називаються полями запису. На відміну від масиву, поля запису можуть бути різних типів. Для реалізації можливості посилання на те чи інше поле, вони мають імена.

Записи подаються у такий спосіб:

```
<ім’я типу> = record
  <ім’я поля>: <тип даних>;
  …
end;
```

наприклад, тип даних, що описує інформацію про студента, може описуватися у такий спосіб:

```
TStudent = record
  name: string[40];
  yearOfBirth: word;
  group, number: string[10];
end;
```

До кожного поля можна отримати доступ, якщо використовувати складне ім'я – вказується ім'я змінної, потім через крапку ім'я поля: student.name.

В пам'яті тип запису займає стільки місця, скільки потрібно для збереження всіх полів запису. Поля запису розташовуються в пам'яті в порядку їхнього подання, наприклад для типу TStudent:

Пам'ять, що займає поле	41 б	2 б	11 б	11 б
Ім'я поля	name	yearOfBirth	group	number

Turbo Pascal дозволяє використовувати записи з так званими варіантними полями, наприклад:

```
var
  mem4 : record
    name : string;
    case Byte of
      0 : (by : array [0..3] of Byte);
      1 : (wo : array [0..1] of Word);
      2 : (lo : longint);
  end;
```

У цьому прикладі тип визначає запис з одним фіксованим полем name та варіантною частиною, котра задається реченням case...of. Варіантна частина складається з декількох варіантів. Кожен варіант визначається константою вибору, за котрою йде двокрапка та список полів, що знаходиться в круглих дужках. В будь-якому записі може бути тільки одна варіантна частина, та, якщо вона є, вона повинна розташовуватися поза всіма фіксованими полями.

Особливістю варіантної частини є те, що всі задані в ній варіанти «накладаються» один на одного, тобто кожному з них виділяється одна й та область пам'яті. Це відкриває додаткові можливості перетворення типів: в прикладі запис mem4 має три варіанти, кожний з яких займає в пам'яті одну й ту ж ділянку з 4 байт. Залежно від того, до котрого поля запису ми звертаємось в програмі, ця ділянка може розглядатися як масив з 4 байт (поле by), масив з двох цілих типу WORD (поле wo), або як одне ціле число типу longint (поле lo). Наприклад, цьому полю можна спочатку присвоїти значення як довгому цілому, а згодом проаналізувати результат за байтами або словами.

Речення case...of, що відкриває варіантну частину, зовнішньо подібне до відповідного оператора вибору, але насправді відіграє роль своєрідного службового слова, що позначає початок варіантної частини. Саме тому в кінці варіантної частини не варто ставить end як пару до case...of. Ключ вибору в реченні case...of фактично ігнорується компілятором: єдина вимога – це те, щоб ключ визначав деякий стандартний або попередньо описаний порядковий тип. Проте цей тип ніяк не впливає на кількість наступних варіантних полів та на характер констант вибору.

– **Множина**

Множини – це набори однотипних логічно пов'язаних між собою об'єктів. Характер зв'язків між об'єктами лише мається на увазі програмістом та ніяк не контролюється Turbo Pascal. Кількість елементів, що входить до множини, може змінюватися в межах від 0 до 256. Саме непостійністю кількості своїх елементів множини відрізняються від масивів та записів.

Дві множини вважаються еквівалентними тоді й тільки тоді, коли всі їхні елементи однакові, причому порядок елементів в множині не має значення. Якщо всі елементи однієї множини входять також і до іншої, говорять про включення першої множини до другої. Порожня множина включається до будь-якої іншої.

Опис типа множини має вигляд:

```
<ім'я типу> = set of <базовий тип>,
```

де <базовий тип> – базовий тип елементів множини, в ролі якого може використовуватися будь-який порядковий тип, крім word, integer, longint.

Для задавання множини використовується так званий конструктор множини: список специфікацій елементів множини, що відділяються один від одного комами; список обрамляється квадратними дужками. Специфікаціями елементів можуть бути константи або вирази базового типу, а також – тип-діапазон того ж базового типу.

Компілятор дозволяє використовувати в ролі базового типу тип-діапазон цілих чисел з мінімальною границею 0 та максимальною 255 або будь-який перерахунковий тип з не більш ніж 256 елементами.

- **Файл**

Під файлом в Turbo Pascal розуміється або іменована область зовнішньої пам'яті ПК (жорсткого диска, гнучкої дискети), або логічний пристрій – потенційне джерело або приймач інформації.

Будь-який файл має три характерні особливості:

– він має ім’я, що відкриває можливість програмі працювати одночасно з декількома файлами;

– він містить компоненти одного типу. Типом компонентів може бути будь-який тип Turbo Pascal, крім файлів;

– розмір файлу, що заново створюється, обмежується тільки ємністю пристроїв зовнішньої пам’яті.

Файловий тип або змінну файлового типу можна задати в один із трьох способів:

```
<ім’я файлового типу> = file of <тип запису>;
<ім’я файлового типу> = text;
<ім’я файлового типу> = file;
```

Залежно від способу визначення можна виділити три типи файлів:

– типізовані файли (задаються реченням file of...);

– текстові файли (визначаються типом text);

– нетипізовані файли (визначаються типом file).

Вид файлу визначає засіб збереження інформації у файлі. Однак в Turbo Pascal немає засобів контролю виду раніше створених файлів. При визначенні вже існуючих файлів програміст повинен сам слідкувати за відповідністю виду характеру визначеного файлу.

- **Вказівники**

Вказівник – це змінна, яка в ролі свого значення містить адресу байта пам’яті. Адресою є два числа, що є номером сегмента пам’яті та номером байта в сегменті. Під кожне число в Turbo Pascal виділяється 2 байти пам’яті.

В мові Pascal виділяються два типи вказівників:

1) типізований – вказівник, що містить адресу ділянки пам’яті з явно вказаним типом даних. Власне вказівник містить адресу тільки першого байта ділянки пам’яті, проте компілятор знає розмір необхідної ділянки. Такі вказівники визначаються за допомогою символу «^», що стоїть перед типом даних, наприклад

```
type
  PInteger = ^Integer;
```

визначає тип-вказівник, що вказує на ділянку пам’яті, яка буде інтерпретуватися як двобайтове ціле число зі знаком;

2) нетипізований – вказівник, що містить адресу першого байта ділянки пам’яті, розмір та інтерпретацію котрої програміст задає у явний спосіб. Визначається за допомогою зарезервованого слова pointer.

- **Об’єкти**

Об’єкт – структура даних, яка містить фіксоване число компонент. Об’єкт є продовженням запису як типу, що об’єднує в собі інформацію різного роду. Проте об’єкт, крім інформації, може містити методи обробки цієї інформації, мати засоби укриття інформації та життєвий цикл змінної об’єктного типу.

Синтаксис визначення:

```
<ім’я типу> = object
  Поле;
  Поле;
  ...
  Метод;
  Метод;
  ...
end;
```

Опис поля об’єкта складається з ідентифікатора, двокрапки й типу даних. Крім того, об’єкт містить заголовки методів. Кожний компонент є або полем (котре містить дані вказаного типу) або методом, котрий виконує операцію з полями об’єкта. Визначення поля містить ідентифікатор, котрий означає поле та його тип даних. Визначення методу містить заголовки процедур, функцій, конструктора або деструктора.

Методи організації обчислювального процесу

Обчислювальний процес в мові Pascal може мати один з чотирьох виглядів: лінійний, розгалужений, ітераційний (циклічний), рекурсивний. Будь-яка програма на верхньому рівні абстракції відповідає лінійному обчислювальному процесу. Більш детальні рівні абстракції розподіляють програму на фрагменти, кожний з яких відповідає одному з чотирьох обчислювальних процесів.

Кожна програма, що написана згідно зі структурною парадигмою програмування, є реалізацією деякого методу обчислень і має декілька рівнів абстракції. Основна програма є найбільш абстрактною реалізацією методу обчислень. За кожним кроком основної програми може стояти більш детальна реалізація, котра оформлюється у вигляді підпрограми, та, в загальному випадку, є самодостатньою.

При організації обчислювального процесу варто дотримуватися такого правила: «Якщо фрагмент коду трапляється в програмі один раз – це гарно, якщо два – це терпимо, а якщо три чи більше – це недопустимо та потрібно винести його у підпрограму».

Лінійний обчислювальний процес

Лінійний обчислювальний процес характеризується виконанням всіх операторів у заданому порядку лише один раз і реалізується шістьома операторами:

1. **Оператор присвоєння** (:=). Використовується у випадку, коли змінній (ліва частина оператора) присвоюється нове значення (права частина оператора). Як ліва частина може виступати:
   - ім’я змінної – правильний ідентифікатор, описаний в розділі var програми чи підпрограми;
   - розадресований вказівник – правильний ідентифікатор, описаний в розділі var програми чи підпрограми як вказівник із символом «^» після ідентифікатора.

Як права частина може виступати будь-який вираз, тип результату котрого сумісний з типом лівої частини. Сумісність типів зазначено в розділі «Операції та стандартні ».

2. **Оператор виклику процедури.** Використовується для виклику підпрограми, що реалізує елементарну дію з точки зору рівня абстракції програми чи підпрограми, де зустрівся виклик процедури, у вигляді підпрограми нижчого рівня абстракції. За кожним викликом процедури (як стандартної, так і користувача) стоїть деяка послідовність дій, що з точки зору основної підпрограми виконується за один крок.

3. **Порожній оператор** (;). Якщо кожен оператор мови Pascal закінчується символом «;», то порожній оператор складається тільки з цього символу. В деяких випадках це є необхідним, наприклад, коли одна з гілок розгалуженого обчислювального процесу чи тіло циклу нічого не виконують.

4. **Складний оператор** (begin…end). Складний оператор є контейнером для деякої послідовності операторів мови Pascal та, з точки зору обчислювального процесу діючого рівня абстракції, виконується як один оператор. Складний оператор використовують в тому випадку, коли потрібно виконати послідовність дій як одну операцію, але ця послідовність трапляється в програмі лише один раз – тобто організація підпрограми не є доцільною. Наприклад:

- коли синтаксис мови Pascal потребує лише одного оператора (як у випадках з умовним оператором та операторами повторень), який має виконати цілу послідовність дій;
- для виділення логічних блоків обчислювального процесу (наприклад, ініціалізація → введення даних → основна логіка програми → виведення даних).

5. **Оператор розгалуження.** Дозволяє залежно від певних умов, що склалися в процесі виконання програми, виконати той чи інший оператор. Оператор розгалуження є основним засобом структурного програмування для створення динамічних програм – таких, що реагують на дії користувача чи іншого середовища. За допомогою оператора розгалуження реалізується розгалужений та рекурсивний обчислювальні процеси.

6. **Оператор повторень.** Дозволяє виконати один і той самий оператор декілька разів. Це основний засіб виконання певної однотипної дії певну кількість разів. За допомогою оператора повторень реалізується ітераційний (циклічний) процес.

Розгалужений обчислювальний процес

Розгалужений обчислювальний процес дозволяє залежно від певних умов, що склалися в процесі виконання програми, виконати той чи інший фрагмент коду. Він характеризується виконанням окремої частини операторів лише один раз. Для організації розгалуженого процесу можуть використовуватися два оператори мови Pascal:

1. **Умовний оператор** дозволяє залежно від певної умови вибрати, який з двох операторів буде виконано. Синтаксис:

```
if <умова> then <оператор1> else <оператор2>;
```

де <умова> – довільний вираз Turbo Pascal логічного типу; <оператор1>, <оператор2> – будь-який оператор Turbo Pascal.

По-перше, обчислюється вираз <умова>, і якщо результат відповідає константі true, виконується <оператор1>, в іншому випадку (константі false) – <оператор2>. Після оператора1 символ «;» не ставиться, тому що, з позиції мови Pascal, умовний оператор – один оператор Pascal.

Гілка else може бути відсутньою, в цьому випадку умовний оператор має скорочений вигляд:

```
if <умова> then <оператор>;
```

У цьому випадку, якщо <умова> відповідає константі true, виконується <оператор>, в іншому випадку – оператор закінчує свою роботу.

2. **Оператор вибору** дозволяє, залежно від ключа вибору, обрати одну з декількох гілок для виконання. Синтаксис:

```
case <ключ_вибору> of
  <константа_вибору1>: <оператор1>;
  <константа_вибору2>: <оператор2>;
  ...
  else <оператор>
end
```

де <ключ_вибору> – довільний вираз будь-якого порядкового типу, <константа_вибору…> – константа того самого типу даних, що й <ключ_вибору>, <оператор…> – будь-який оператор Turbo Pascal.

Спочатку обчислюється вираз <ключ_вибору>, потім отримане значення порівнюється з кожною із констант вибору в тому порядку, як вказано в програмі, від першого до останнього. Якщо обчислене значення співпадає з однією з констант, виконується оператор, що відповідає константі, і далі перевірка не виконується. Тобто, якщо в константах вибору є дві однакові константи, виконується оператор, що відповідає тільки першій.

Якщо результат обчислення виразу <ключ_вибору> не збігся з жодного з констант вибору, виконується гілка else, якщо вона присутня. Якщо потрібно виконати однаковий оператор по різним константам, допустимо вказати їх через кому.

Розгалужений обчислювальний процес – це головний засіб створення динамічних програм, що реагують на оточення. За допомогою розгалуженого процесу забезпечується динамічний інтерфейс користувача.

Ітераційний (циклічний) обчислювальний процес

Ітераційний процес обчислень дозволяє виконати одну елементарну дію певну кількість разів. Кожне виконання цієї дії називається ітерацією. Для органі-

зації ітераційного процесу можуть використовуватися три оператори циклу мови Pascal:

1. **Цикл з параметром** дозволяє виконати певну дію певну кількість разів. Використовується в тому випадку, коли відома кількість виконань оператора. Синтаксис:

```
for <параметр> := <початкове значення> to <кінцеве значення> do
  <оператор>;
```

```
for <параметр> := <початкове значення> downto <кінцеве значення> do
  <оператор>;
```

де <параметр> – змінна будь-якого порядкового типу даних, <початкове значення> та <кінцеве значення> – вирази того ж самого типу даних, що й <параметр>, <оператор> – будь-який оператор мови Pascal.

Оператор працює у такий спосіб:

1) обчислюється вираз <початкове значення> та результат присвоюється змінній <параметр>;

2) обчислюється вираз <кінцеве значення>, якщо результат менше (якщо використовувалася перша форма запису – зі словом to) чи більше (якщо використовувалася друга форма запису – зі словом downto), ніж значення змінної <параметр>, то оператор закінчує свою роботу;

3) виконується <оператор>;

4) значення змінної <параметр> на +1 (якщо використовувалася перша форма запису – зі словом to) чи на –1 (якщо використовувалася друга форма запису – зі словом downto);

5) виконується перехід до кроку 2.

Змінна <параметр> доступна всередині оператора циклу, проте змінювати її значення всередині оператора циклу, згідно зі структурною парадигмою програмування, неприпустимо, але припустимо використовувати її як параметр, що не змінюється, наприклад, як індекс масиву.

2. **Оператор циклу з попередньою перевіркою умови** дозволяє виконувати певний оператор до тих пір, доки виконується певна умова. Синтаксис оператора:

```
while <умова> do <оператор>;
```

де <умова> – будь-який вираз мови Pascal, що має логічний тип результату, <оператор> – будь-який оператор мови Pascal.

Оператор працює у такий спосіб:

1) обчислюється вираз <умова> так, якщо результат відповідає константі false, припиняє роботу оператора циклу;

2) виконується <оператор>, після чого здійснюється перехід до пункту 1.

Необхідно слідкувати, щоб значення виразу <умова> могло змінюватися під час виконання оператора циклу, інакше цикл може стати нескінченним.

3. **Оператор циклу з остаточною перевіркою умови** дозволяє виконувати певний оператор до тих пір, доки виконується певна умова. Синтаксис оператора:

```
repeat <тіло циклу> until <умова>;
```

де <умова> – будь-який вираз мови Pascal, що має логічний тип результату, <тіло циклу> – послідовність будь-яких операторів мови Pascal.

Оператор працює у такий спосіб:

1) виконуються всі оператори, що складають <тіло циклу> у заданому порядку;

2) обчислюється вираз <умова> та, якщо результат відповідає константі true, припиняє роботу оператора циклу, інакше – здійснюється перехід до пункту 1.

Необхідно слідкувати, щоб значення виразу <умова> могло змінюватися під час виконання оператора циклу, інакше цикл може стати нескінченним.

Усередині циклу допустимо використовувати два спеціальні оператори:

1) **break** – достроково припиняє виконання оператора циклу. У випадку, коли оператор використовується в циклі з параметром (for), параметр циклу зберігає своє останнє значення, тобто можна визначити, на котрій ітерації цикл припинив свою роботу;

2) **continue** – достроково припиняє виконання поточної ітерації та переходить до наступної операції.

Рекурсивний обчислювальний процес

Рекурсивний обчислювальний процес дозволяє реалізовувати практично всі математичні алгоритми. Більшість математичних алгоритмів є рекурсивними, тобто визначаються через самих себе. Деякі алгоритми не мають прийнятної ітераційної форми. Реалізація рекурсивних алгоритмів у вигляді рекурсивних програм, звичайно, є простою, читабельною та красивою. Крім того, обробка рекурсивних об’єктів (наприклад, об’єкти файлової системи) також примушує використовувати рекурсивні процедури. Проте рекурсія має свої суттєві недоліки.

Реалізація рекурсивного обчислювального процесу виконується шляхом визначення підпрограми, яка прямо чи побічно викликає себе. Прикладом рекурсивної підпрограми є реалізація функції Акермана:

```
function Ackermann(const n, m: comp): comp;
begin
  if (m = 0) then
     Ackermann := n+1
  else if (m > 0) then
    if (n = 0) then
       Ackermann := Ackermann(m-1, 1)
    else if (n > 0) then
       Ackermann := Ackermann(m-1,  Ackermann(m, n-1));
end;
```

Рекурсивний обчислювальний процес може характеризується такими показниками:

1) за положенням рекурсивного виклику:
   а) висхідна – рекурсивний виклик трапляється на початку підпрограми;
   б) низхідна – рекурсивний виклик трапляється наприкінці підпрограми;
   в) комбінована – рекурсивний виклик трапляється всередині підпрограми;
2) за типом рекурсивного виклику:
   а) пряма – підпрограма викликає сама себе явно;
   б) побічна – підпрограма викликає іншу підпрограму, яка, в свою чергу, викликає першу підпрограму;
3) за складністю рекурсивного виклику:
   а) лінійна – у кожному рекурсивному операторі (виклику) трапляється тільки один рекурсивний виклик;
   б) нелінійна – принаймні в одному рекурсивному операторі (виклику) трапляється більш ніж один рекурсивний виклик (приклад – функція Акермана). Те саме стосується і складних операторів: наприклад, коли в операторі циклу зустрічається рекурсивний виклик – рекурсія нелінійна.

Рекурсивні підпрограми мають такі недоліки:

- виклик підпрограми супроводжується передаванням керування в інший фрагмент коду зі збереженням адреси виклику в стеку, а при виході з підпрограми – знову керування передається в інший фрагмент коду за збереженої адреси. Тобто, кожний виклик процедури займає деякий час. Отже, зазвичай, рекурсивні підпрограми працюють довше, ніж ітераційні підпрограми, при реалізації однакового методу обчислень;

- виклик підпрограми супроводжується передаванням вхідних та вихідних значень через стек програми. Локальні змінні також розміщуються в сегменті стеку. Отже, особливо за нелінійної рекурсії, можливе переповнення стеку, що призводить до аварійної зупинки програми;
- побічна рекурсія, зазвичай, не є очевидною, тому складна для сприйняття та є потенційним джерелом логічних помилок, таких як нескінченна рекурсія.

Оскільки в багатьох випадках все ж таки рекурсія є доцільною, то, враховуючи недоліки, при організації рекурсивного обчислювального процесу програмісту варто дотримуватися таких правил:

- якщо реалізація рекурсивного методу обчислень не є очевидною та легкою для сприйняття, та існує аналогічний за складністю та результатом ітераційний метод обчислень, варто відмовитися від неї та використовувати ітераційний метод обчислень – гірше не буде;
- варто уникати побічної та нелінійної рекурсії як потенційного джерела помилок;
- варто мінімізувати кількість та розмір параметрів, що передаються у підпрограму та повертаються з неї, а також кількість та розмір локальних змінних, задля уникнення проблеми переповнення стеку. Мінімізація параметрів досягається шляхом передачі в підпрограми параметрів-змінних чи параметрів-констант, а мінімізація локальних змінних досягається шляхом використання глобальних змінних, якщо це можливо;
- варто уникати рекурсивних викликів всередині операторів циклу, крім випадків, коли обробляються специфічні рекурсивні об’єкти, такі як дерево каталогів.

Директиви компілятора

Директиви компілятора – це спеціальні послідовності символів, що використовуються при компіляції програми для визначення умов компіляції. В результуючій програмі ці коди не присутні та ніяк не впливають на хід обчислювального процесу вже скомпільованої програми. Нижче наведено список всіх директив компілятора середовища Turbo Pascal.

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$A+} або {$A-}	{$A+}	Глобальна	Options|Compiler|Align Data
Опис			
Переключає між вирівнюванням за байтами та за словами змінних та типізованих констант.			
Принцип дії			
Вирівнювання за словами не дає ефекту на процесорах 80x88. На всіх процесорах 80x86 вирівнювання по словам задає більш швидке виконання. – До даних розміру «слово» (2 байти) за парною адресою можна звернутися за один цикл пам’яті. – До даних за непарними адресами можна звернутися за два цикли пам’яті. У стані $A+ всі змінні й типізовані константи розміром більше 1-го байта розташовуються по парним адресам. За необхідності, між змінними можуть бути додані байти, що не використовуються для досягнення вирівнювання за словами. Режим $A+ не впливає на змінні розміром 1 байт, поля записів або елементи масивів. Поле в запису буде вирівняно за словами тільки в тому випадку, якщо повний розмір всіх полів до нього є парним. Щоб кожен елемент масиву було вирівняно за словами, розмір елементів масиву повинен бути парним. У стані $A- вирівнювання не виконується. Змінні та типізовані константи лише розміщуються в наступну доступну адресу пам’яті, незалежно від їхнього розміру. Незалежно від стану $A, кожний розділ опису глобальних змінних та констант завжди починається на межі слів. Компілятор завжди зберігає вказівник вершини стека (SP) вирівняним за словами, розподіляючи додатковий невикористаний байт у запису активації процедури, якщо це потрібно.			

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$B+} або {$B-}	{$B-}	Локальна	Options|Compiler|Boolean Evaluation

Опис

Перемикає між двома різними моделями генерації об'єктного коду для булевих операторів AND та OR.

Принцип дії

У стані $B+ компілятор генерує код для повної булевої оцінки виразу. Це означає, що кожний операнд булевого виразу, створеного за допомогою операторів AND та OR, буде оцінено навіть у тому випадку, якщо результат всього виразу вже відомий.

У стані $B- компілятор генерує код з короткою схемою оцінки булевого виразу. Це означає, що оцінка виразу зупиняється, як тільки результат всього виразу стає очевидним.

Синтаксис	Значення за замовчуванням	Тип
{$C атрибут атрибут}	{$C MOVEABLE DEMANDLOAD DISCARDABLE}	Глобальна

Опис

Управляє атрибутами сегмента коду.

Принцип дії

Кожен сегмент коду в додатку або бібліотеці має набір атрибутів, котрі визначають його поведінку, коли він завантажується в пам'ять.

Директива $C впливає тільки на сегмент коду модуля, програми або бібліотеці, в яких вона розміщена. Атрибути сегмента коду розділено на групи по два в кожній. Кожна опція має протилежну їй опцію:

MOVEABLE FIXED	Система може змінювати розташування сегмента коду в пам'яті. Система не може змінювати розташування сегмента коду в пам'яті.
PRELOAD DEMANDLOAD	Сегмент коду завантажується при старті програми. Сегмент коду завантажується тільки за необхідності.
PERMAMENT DISCARDABLE	Сегмент коду залишається в пам'яті завжди після завантаження. Сегмент коду може бути вивантажено з пам'яті, якщо він більше не потрібен.

Якщо задані дві опції, тільки остання буде мати значення.

Наприклад, запис {$C FIXED MOVEABLE DISCARDABLE} створює сегмент коду з атрибутами MOVEABLE и DISCARDABLE.

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$D+} або {$D-}	{$D+}	Глобальна	Options|Compiler|Debug Information

Опис

Вмикає або вимикає генерацію відлагоджувальної інформації.

Принцип дії

Відлагоджувальна інформація складається з таблиці номерів рядків для кожної процедури. В таблиці записано відповідність адрес об'єктного коду номерам рядків похідного тексту. Коли відлагоджувальну інформацію включено для цієї програми або модуля, можна використовувати налагоджувальники Borland для покрокового виконання та встановлення точок переривання в цьому модулі. Якщо в модулі або програмі, що компілюється в стані {$D+}, трапляється помилка періоду виконання, Turbo Pascal автоматично переходить до оператора, котрий викликав помилку. Перемикач Options |Linker| Map File виробляє інформацію для цього модуля тільки в тому випадку, якщо він компілювався в стані $D+. Для модулів відлагоджувальна інформація записується в TPW-, TPP- або TPU-файл разом з об'єктним кодом. Відлагоджувальна інформація збільшує розмір TPU-, TPW- і TPP-файлів та займає додаткове місце в пам'яті при компіляції програм, що використовує модуль, але вона не впливає на розмір або швидкість виконання програми. Перемикач Debug Information зазвичай використовується разом з перемикачем Local Symbols.

Синтаксис

{$DEFINE Ім'я}

Опис

Визначає умовний символ із заданим іменем

Принцип дії

Певний символ існує до кінця компіляції, або доки його не буде видалено директивою $UNDEF Ім'я.

Директива {$DEFINE Ім'я} не дає ефекту, якщо символ з іменем «Ім'я» вже було визначено.

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$E+} або {$E-}	{$E+}	Глобальна	Options|Compiler|Emulation

Опис

Вмикає або вимикає компоновку бібліотеки програмної емуляції 80x87 в EXE-файл.

Принцип дії

Директива $E вмикає або вимикає компоновку бібліотеки емуляції математичного сопроцесора 80x87 за його відсутності.

Коли програма компілюється в стані {$N+, E+}, компілятор компонує програму з повним емулятором 80x87. Створений в результаті цього EXE файл

може бути виконано на будь-якій машині, незалежно від того, має вона сопроцесор чи ні. Якщо сопроцесор присутній, то програма використовує його, інакше сопроцесор емулюється відповідною бібліотекою підтримки.

У стані {$N+, E-} компілятор компонує програму з меншою бібліотекою підтримки чисел з рухомою крапкою, котра може використовуватися тільки в тому випадку, якщо під час виконання програми є сопроцесор. Стан перемикача {$E} не має ніякого значення, якщо він використовуватися в модулі. Цей перемикач використовується тільки при компіляції програм.

Якщо програма компілюється в стані {$N-} та всі модулі, що використовуються в програмі, компілювалися в стані {$N-}, то бібліотека підтримки 80x87 непотрібна, та стан перемикача {$E} ігнорується.

Синтаксис
{$ELSE}
Опис
Компілює або пропускає текст після директиви $ELSE.
Принцип дії
Всередині тексту програми, розділеного директивами $IFDEF (або $IFNDEF) та $ENDIF, $ELSE компілює текст, що йде після $ELSE, якщо умова $IFDEF (або $IFNDEF) не виконується. Якщо умова $IFDEF (або $IFNDEF) виконується, то $ELSE ігнорує блок тексту, наступний за словом $ELSE.

Синтаксис
{$ENDIF}
Опис
Завершує блок умовної компіляції, що починається з останньої умовної директиви $IFxxx.

Синтаксис	**Значення за замовчуванням**	**Тип**	**Команда меню**
{$F+} або {$F-}	{$F-}	Локальна	Options|Compiler|Force Far Calls
Опис			
Задає модель виклику, яка використовується для процедур та функцій, що послідовно компілюються.			
Принцип дії			
Процедури та функції, що компілюються в стані $F+, завжди використовують модель дальнього (FAR) виклику. У стані $F-, компілятор автоматично вибирає відповідну модель: – FAR, якщо процедура або функція заявлена в розділі інтерфейсу модуля; – NEAR, якщо процедура або функція заявлена в іншому місті. Для програм, що використовують оверлеї, директива $F+ на початку програми та кожного модуля необхідна. Для програм, що використовують процедурні змінні, всі ці процедури мають використовувати далеку (FAR) модель коду.			

Синтаксис	Тип
{$G Ім’я модуля, Ім’я модуля ...}	Локальна
Опис	
Визначає групу модулів, які компонувальник повинен помістити в один сегмент пам’яті.	
Принцип дії	
Групування модулів в одному сегменті гарантує, що модулі завантажуються та вивантажуються з пам’яті в один і той самий час. Директива $G використовується для групування модулів, що містять частини коду, які в програмі не використовуються. Кожна директива $G визначає групу модулів. Директиви $G допустимі тільки в програмі або бібліотеці та повинні знаходиться після зарезервованого слова uses. Компілятор сповіщає про помилку, якщо модуль додано в декілька груп. В доповнення до будь-яких груп, що створені за допомогою директиви $G, компілятор підтримує групу за замовчуванням, котра включає всі модулі, що згруповані неявно. Компонувальник мінімізує кількість сегментів коду в файлі для виконання, об’єднуючи всі модулі, що належать одній групі	

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$I+} або {$I-}	{$I+}	Локальна	Options|Compiler|I/O checking
Опис			
Вмикає або вимикає автоматичну генерацію об’єктного коду, котра перевіряє результат звернення до процедур введення / виведення.			
Принцип дії			
Якщо процедура введення / виведення повертає ненульовий результат введення / виведення, коли перемикач $I ввімкнено, то програма завершується та виводиться повідомлення про помилку періоду виконання. Якщо перемикач $I вимкнено, то можливе використання функції IOResult задля перевірки помилок введення / виведення.			

Синтаксис	Тип
{$I Ім’я файлу}	Локальна
Опис	
Наказує компілятору додати зазначений файл в компіляцію.	
Принцип дії	
Файл вставляється в текст, що компілюється, відразу після директиви $I. Можливі до 15 рівнів вкладених файлів. Файл для додавання не може бути визначено всередині операторної частини. Всі оператори між словами begin и end операторної частини повинні знаходитися в одному файлі.	

Синтаксис
{$IFDEF Ім’я}
Опис
Компілює текст, що йде за директивою IFDEF, якщо символ з іменем «Ім’я» визначено.

Синтаксис
{$IFNDEF Ім’я}
Опис
Компілює текст, що йде за директивою IFNDEF, якщо символ з іменем «Ім’я» не визначено.

Синтаксис
{$IFOPT Перемикач}
Опис
Компілює текст, наступний за директивою IFOPT, якщо заданий перемикач знаходиться в певному стані.
Принцип дії
Параметр «Перемикач» додає ім’я перемикача, що супроводжується знаком «+» або «–».

Синтаксис	**Значення за замовчуванням**	**Тип**	**Команда меню**
{$L+} або {$L-}	{$L+}	Глобальна	Options|Compiler|Local symbols
Опис			
Вмикає або вимикає генерацію інформації про локальні символи.			
Принцип дії			
Інформація про локальні символи складається з: – символів в частині реалізації модуля (імен та типів всіх локальних змінних і констант модуля), – символів всередині процедур та функцій модуля. Коли опцію «Local symbols» ввімкнено, можна використовувати налагоджувальник задля перевірки та зміни значень локальних змінних модуля або програми. Для модулів інформація про локальні символи записується у файл модуля разом з об’єктним кодом модуля. Інформація про локальні символи збільшує розмір файлу модуля та займає додаткову область пам’яті при компіляції програм, що використовують модуль. Вона не впливає на розмір або на швидкість виконання готової програми. Цей перемикач звичайно використовується разом з перемикачем відлагоджувальної інформації $D. Директива $L ігнорується, якщо відлагоджувальну інформацію вимкнено {$D-}.			

Синтаксис	Тип	Команда меню
{$M розмір стеку, розмір кучі} (Windows) {$M розмір стеку} (Захищений режим DOS) {$M розмір стеку, початок кучі, кінець кучі} (Реальний режим DOS)	Глобальна	Options|Compiler| Memory sizes
Опис		
Визначає параметри розподілення пам'яті.		
Принцип дії		
Директива $M визначає параметри розподілення пам'яті для програми або бібліотеки. Параметр «розмір стеку» повинен бути цілим числом в діапазоні від 1 024 до 65 520, що визначає розмір сегмента стека. Для реального режиму DOS параметри «початок кучі» та «кінець кучі» визначають мінімальний та максимальний розміри кучі відповідно. Параметр «початок кучі» повинен бути в діапазоні від 0 до 655 360, а параметр «кінець кучі» повинен бути в діапазоні від «початок кучі» до 655 360. Для Windows параметр «розмір кучі» визначає розмір локальної області кучі у сегменті даних. Параметр «розмір кучі» повинен бути цілим числом в діапазоні від 0 до 65 520. Директива $M не дає ефекту, коли використовується в модулі. Параметр «розмір стеку» ігнорується в бібліотеці.		

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$N+} або {$N-}	{$N-}	Глобальна	Options|Compiler|8087/80287
Опис			
Перемикає між двома різними моделями генерації об'єктного коду обробки чисел с рухомою крапкою, що забезпечуються компілятором.			
Принцип дії			
У стані $N- компілятор генерує код для виконання всіх дійсних обчислень програмно, викликаючи підпрограми бібліотеки підтримки. У стані $N+ компілятор генерує код для виконання всіх дійсних обчислень з використанням математичного сопроцесора 80x87 і надає доступ до чотирьох дійсних типів: single, double, extended та comp. Також можливе використання директиви $E+, щоб емулювати 80x87. Це надає доступ до дійсних типів за відсутності 80x87 сопроцесора.			

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$O+} або {$O-}	{$O-}	Глобальна	Options|Compiler|Overlays allowed
Опис			
Вмикає та вимикає генерацію оверлейного об'єктного коду.			
Принцип дії			
У стані {$O+} генератор об'єктного коду приймає спеціальні засоби при передаванні рядкових та набірних констант з однієї оверлейної процедури або			

функції в іншу. Використання директиви {$O+} в модулі не робить його оверлейним, але компілятор дозволяє модулю бути оверлейним тільки в тому випадку, якщо він компілювався в режимі {$O+}.

Якщо модуль планується використовувати і в оверлейних, і в неоверлейних програмах, то компілюйте їх в режимі {$O+}, що гарантує Вам можливість використовувати один й той самий модуль в різних типах програм. Директива компілятора {$O} майже завжди використовується з директивою {$F+}, щоб задовольнити вимогу дальнього (FAR) типу виклику менеджера оверлеїв.

Синтаксис	**Тип**
{$O Ім’я модуля}	Локальна
Опис	
Записує модуль у файл оверлеїв.	
Принцип дії	
При компіляції програми директива {$O Ім’я модуля} визначає, який з модулів, що використовуються програмою, повинен бути поміщений в OVR файл замість EXE файлу. Директиви {$O Ім’я модуля} повинні знаходитися після розділу uses програми. Директива {$O Ім’я модуля} не дає ефекту, якщо використовується в модулі. Будь-який модуль, що описано в директиві {$O Ім’я модуля}, повинен компілюватися із ввімкненою опцією Overlays allowed ($O+).	

Синтаксис	**Значення за замовчуванням**	**Тип**	**Команда меню**
{$P+} або {$P-}	{$P-}	Глобальна	Options|Compiler|Open parameters
Опис			
Керує значенням змінних параметрів, що заявлено з використанням зарезервованого слова string.			
Принцип дії			
У стані $P- змінні параметри, що заявлено за допомогою ключового слова string, є нормальними змінними параметрами. Це забезпечує сумісність з більш ранніми версіями Turbo Pascal. У стані $P+ змінні параметри, що заявлено за допомогою ключового слова string, є відкритими рядковими параметрами. Незалежно від стану директиви $P, для визначення відкритих рядкових параметрів може використовуватися ідентифікатор OpenString. Фактичним параметром відкритого рядкового параметра може бути змінна будь-якого рядкового типу, та, всередині процедури або функції, атрибут максимальної довжини формального параметра буде таким самим, як і у фактичного параметра.			

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$Q+} або {$Q-}	{$Q-}	Локальна	Options|Compiler|Overflow checking
Опис			
Керує генерацією коду перевірки переповнення.			
Принцип дії			
У стані $Q+ деякі цілочисельні операції перевіряються на переповнення, наприклад: +, –, *, abs, sqr, succ та pred. Код для кожної з цих арифметичних операцій супроводжується додатковим кодом, котрий перевіряє, чи знаходиться результат всередині діапазону, що забезпечується типом даних. Якщо перевірка переповнення не витримується, то програма завершується та виводиться повідомлення про помилку періоду виконання програми. {$Q} не впливає на стандартні процедури inc та dec. Ці процедури ніколи не перевіряються на переповнення. Перемикач $Q зазвичай використовується разом з перемикачем $R. Ввімкнення перевірки переповнення затримує програму та робить її більше, тобто перемикач {$Q+} доцільно використовувати тільки для відлагоджування. В стані $Q- перевірка переповнення не виконується.			

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$R+} або {$R-}	{$R-}	Локальна	Options|Compiler|Range Checking
Опис			
Вмикає та вимикає генерацію коду перевірки діапазону.			
Принцип дії			
У стані $R+: – усі індекси масивів та рядків перевіряються на приналежність до припустимих меж; – усі присвоювання скалярним та піддіапазонним змінним перевіряються на приналежність заданим діапазонам. Якщо перевірка приналежності до діапазону не витримується, то програма завершується та виводиться повідомлення про помилку періоду виконання програми. $R+ не впливає на процедури inc та dec. Ввімкнення перевірки діапазону затримує програму та робить її більше, тобто перемикач {$R+} доцільно використовувати тільки для відлагоджування.			

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$S+} або {$S-}	{$S+}	Локальна	Options|Compiler|Stack checking
Опис			
Вмикає та вмикає генерацію коду перевірки переповнення стеку.			
Принцип дії			
У стані $S+ компілятор генерує спеціальний код на початку кожної процедури або функції для перевірки: чи достатньо місця в стеку для локальних змінних тієї чи іншої тимчасової пам’яті. Коли в стеку недостатньо вільного			

місця, виклик процедури або функції, що компілюються в режимі $S+, примушує програму завершитися та вивести повідомлення про помилку періоду виконання програми.

У стані $S-, коли в стеку недостатньо вільного місця, виклик процедури або функції, певно, спровокує збій системи.

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$T+} або {$T-}	{$T-}	Локальна	Options|Compiler|Typed @ operator
Опис			
Керує типами вказівників, що створені оператором @.			
Принцип дії			
У стані $T- тип результату оператора @ – завжди нетипизований вказівник. Коли оператор @ застосовується до змінної в стані $T+, тип результату – ^T, де T сумісний тільки з іншими вказівниками на заданий тип змінної.			

Синтаксис
{$UNDEF Ім’я}
Опис
Прибирає попередньо визначений умовний символ з іменем «Ім’я».
Принцип дії
Символ забувається до кінця компіляції або доки його знову не буде визначено за допомогою директиви $DEFINE. Директива {$UNDEF Ім’я} не дає ефекту, якщо символ з іменем «Ім’я» вже видалено.

Синтаксис	Значення за замовчуванням	Тип	Команда меню
{$V+} або {$V-}	{$V+}	Локальна	Options|Compiler|Strict var-strings
Опис			
Керує перевіркою типів рядків, що передані як змінні параметри.			
Принцип дії			
У стані $V+ виконується строгий контроль відповідності типів, щоб формальні та фактичні параметри мали ідентичні рядкові типи. У стані $V- будь-яка змінна з рядковим типом дозволяється як фактичний параметр, навіть якщо заявлена максимальна довжина не така ж сама, як і у формального параметра.			

Синтаксис	За замовчуванням	Тип	Команда меню
{$W+} або {$W-}	{$W+}	Локальна	Options|Compiler|Windows stack frames
Опис			
Генерує спеціальні пролог- та епілог-коди для дальніх процедур та функцій.			

Принцип дії

У стані $W+ компілятор генерує спеціальний код входу та виходу для дальніх процедур та функцій. Цей код дає можливість менеджеру пам'яті реального режиму Windows правильно ідентифікувати дальні записи активації, коли він корегує ланцюжок викликів після переміщення сегмента коду або даних. В стані $W- компілятор не генерує такого коду.

Синтаксис	За замовчуванням	Тип	Команда меню
{$X+} або {$X-}	{$X+}	Локальна	Options|Compiler|Extended syntax

Опис

Вмикає або вимикає розширений синтаксис Turbo Pascal.

Принцип дії

У стані $X+ функції можуть використовуватися як звичайні оператори – результат функції може бути відкинуто. Взагалі, обчислення, що виконуються функцією, представляються її результатом, проте, в деяких випадках функція може виконувати декілька операцій, ґрунтуючись на параметрах. В деяких з цих випадків вона може не виробляти результат. В таких випадках, $X+ дозволяє обробляти функцію як процедуру. Директива $X+ не належить до вбудованих функцій (функцій, що визначені в модулі System). Режим $X+ також вмикає підтримку рядків що закінчуються нулем, активізуючи спеціальні правила, що належать до вбудованого типу PChar та нуль-заснованим символьним масивам.

У стані $X- розширений синтаксис вимкнено. Спроба його використати призведе до помилки.

Синтаксис	За замовчуванням	Тип	Команда меню
{$Y+} або {$Y-}	{$Y+}	Локальна	Options|Compiler|Symbol information

Опис

Вмикає або вимикає генерацію інформації про посилання на символи.

Принцип дії

Інформація про посилання на символи складається з таблиць, в котрих знаходяться номери рядків всіх визначень та посилань на символи модуля.

Стан $Y+ вмикає генерацію інформації про посилання на символи. Інформація про посилання на символи для модулів записується в TPU-, TPP- або TPW-файли разом з об'єктним кодом модуля. Інформація про посилання на символи збільшує розмір TPU-, TPP- або TPW-файлів, але не впливає на розмір або швидкість готової програми, що використовує цей модуль. Перемикач $Y не дає ефекту, якщо вимкнені перемикачі $D и $L.

У стані $Y- генерація інформації про посилання на символи вимкнена.

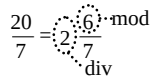
Операції та стандартні підпрограми

Операції

Операції в мові Pascal є основним засобом формування виразів. Вираз може складатися з: операцій, їхніх операндів, дужок та викликів функцій. Порядок обчислення виразу є таким за наведеним порядком:

1. По-перше, обчислюється та частина виразу, що знаходиться в дужках.
2. Якщо одним з операндів деякої операції є виклик функції, то спочатку здійснюється виклик функції, а потім – операція.
3. Операції виконуються згідно з пріоритетом за таким порядком:
   1) унарні операції: not, @, – (унарний);
   2) мультиплікативні операції: *, /, div, mod, and, shl, shr;
   3) адитивні операції: +, – (бінарний), or, xor;
   4) операції відносин: =, <>, <, >, <=, >=, in.
4. Порядок виконання декількох операцій одного пріоритету встановлюється компілятором за умови оптимізації коду програми і не обов'язково зліва направо.
5. При обчисленні логічних виразів одного пріоритету їхнє виконання завжди зліва направо, причому будуть обчислені всі або тільки достатні операції, що залежить від директиви $B або, за її відсутності, опції Options | Compiler | Boolean Evaluation.

Операція	**Тип операндів**	**Тип результату**	**Дія**	**Приклади**
not	логічний	логічний	логічне заперечення	not true → false
not	будь-який цілий	такий саме, як і операнд	побітове заперечення	not 27 (00011011) → 228 (11100100)
@	будь-який	вказівник (залежно від директиви $t – типізований або нетипізований)	отримання адреси розташування в пам'яті	p:=@a → в змінній p – вказівник на розташування значення змінної a в пам'яті
*	будь-який цілий	найменший цілий, в котрий поміщається результат	множення	2 (byte) * 200 (byte) → 400 (integer)
*	будь-який дійсний	extended, якщо {$n+}, та real, якщо {$n–}	множення	2.0 * 200.0 → 400.0

Операція	Тип операндів	Тип результату	Дія	Приклади
*	множинний	множинний	перетин множин: результат містить елементи, що є в обох множинах	[1, 2, 3, 4, 5] * [4, 5, 6, 7] → [4, 5]
/	будь-який дійсний	extended, якщо {$n+}, та real, якщо {$n–}	ділення	2.0 / 200.0 → 0.01
div	будь-який цілий	найменший цілий, в котрий поміщається результат	цілочисельне ділення 	1 div 2 → 0 20 div 7 → 2
mod	будь-який цілий	найменший цілий, в котрий поміщається результат	залишок від ділення. Увага! Залишок від ділення – завжди ціле число, що не менше за 0 та менше за дільник!	1 mod 2 → 1 20 mod 7 → 6
and	логічний	логічний	логічне і	true and false → false
and	будь-який цілий	такий самий, що і операнд	побітове і	27 (00011011) and 204 (11001100) → 8 (00001000)
shl	будь-який цілий	найменший цілий, в котрий поміщається результат	лівий зсув	27 (00011011) shl 2 → 6 (00000110)
shr	будь-який цілий	найменший цілий, в котрий поміщається результат	правий зсув	27 (00011011) shl 2 → 108 (01101100)
+	будь-який цілий	найменший цілий, в котрий поміщається результат	додавання	100 (byte) + 200 (byte) → 300 (integer)
+	будь-який дійсний	extended, якщо {$n+}, та real, якщо {$n–}	додавання	100.0 + 200.0 → 300.0
+	множинний	множинний	об’єднання множин: результат містить всі елементи першої та другої множин	[1, 2, 3, 4, 5] * [4, 5, 6, 7] → [1, 2, 3, 4, 5, 6, 7]

Операція	Тип операндів	Тип результату	Дія	Приклади
+	рядковий	рядковий	з'єднання рядків	«turbo » + «pascal» → «turbo pascal»
–	будь-який цілий	найменший цілий, в котрий поміщається результат	віднімання	300 – 100 → 200
–	будь-який дійсний	extended, якщо {$n+}, та real, якщо {$n–}	віднімання	5.1 – 1.3 → 3.8
or	логічний	логічний	логічне або	true or false → true
or	будь-який цілий	такий самий, що і операнд	побітове або	27 (00011011) or 204 (11001100) → 223 (11011111)
xor	логічний	логічний	логічне виключне або	true xor false → true
xor	будь-який цілий	такий самий, що і операнд	побітове виключне або	27 (00011011) or 204 (11001100) → 215 (11010111)
=	базові типи даних та масиви	логічний	перевірка на рівність	'a' = 'a' → true 12 = 10 → false
=	множинний	логічний	перевірка на еквівалентність	[1, 2, 3] = [2, 1, 3] → true
<>	базові типи даних та масиви	логічний	перевірка на нерівність	'a' <> 'a' → false 12 <> 10 → true
<>	множинний	логічний	перевірка на нееквівалентність	[1, 2, 3] = [2, 1, 4] → true
<	будь-який базовий	логічний	перевірка на менше	'a' < 'a' → false 10 < 12 → true
<=	будь-який базовий	логічний	перевірка на менше або дорівнює	'a' <= 'a' → true 12 <= 10 → true
<=	множинний	логічний	перевірка на входження	[1, 2, 3] <= [1, 2, 3, 4, 5] → true
>	будь-який базовий	логічний	перевірка на більше	'a' > 'a' → false 12 > 10 → true
>=	будь-який базовий	логічний	перевірка на більше або дорівнює	'a' >= 'a' → true 12 >= 10 → true
>=	множинний	логічний	перевірка на входження	[0, 1, 2, 3, 4] >= [1, 2, 3] → true

Операція	Тип операндів	Тип результату	Дія	Приклади
in	праворуч – множинний, ліворуч – базовий тип для множини	логічний	перевірка на входження окремого елемента в множину	1 in [1, 2, 3] → true

Існує декілька правил автоматичного перетворення типів даних, згідно з якими, замість операндів, що наведені в таблиці, можливе використання операндів іншого типу даних. Ці правила визначають сумісність типів. Припустимо, що T1 – тип змінної, а T2 – тип виразу та виконується оператор присвоєння: T1:= T2. Це присвоєння можливе тільки у таких випадках:

– T1 та T2 є одним і тим самим типом, проте цей тип не належить до файлів або структурованих типів, що містять файли;

– T1 та T2 є сумісними порядковими типами (обидва є одним і тим самим типом, або обидва – цілі, або один є діапазоном іншого), та значення T2 знаходиться в діапазоні можливих значень T1;

– T1 та T2 є дійсними типами, та значення T2 знаходиться в діапазоні можливих значень T1;

– T1 – дійсний тип, та T2 – цілий тип;

– T1 – рядок, та T2 – символ;

– T1 – рядок та T2 – упакований рядок;

– T1 та T2 – упаковані рядки однакової максимальної довжини;

– T1 та T2 – сумісні множини (складені з елементів одного базового типу) та всі члени T2 належать множині можливих значень T1;

– T1 та T2 – сумісні вказівники (типізовані вказівники на сумісні типи даних, або T1 – нетипізований вказівник, та T2 – будь-який типізований вказівник, або навпаки);

– T1 та T2 – сумісні процедурні типи (мають однаковий тип результату, кількість параметрів, тип взаємно відповідних параметрів);

– T1 – об’єкт, та T2 – його нащадок.

Мова Turbo Pascal є сильно типізованою, завдяки цьому ця мова більш проста для програмістів-початківців. Проте, існує багато засобів перетворити один тип даних на інший.

Логічні операції діють за такими правилами:

X	Y	not X	X and Y	X or Y	X xor Y
0 (false)	0 (false)	1 (true)	0 (false)	0 (false)	0 (false)
0 (false)	1 (true)	1 (true)	0 (false)	1 (true)	1 (true)
1 (true)	0 (false)	0 (false)	0 (false)	1 (true)	1 (true)
1 (true)	1 (true)	0 (false)	1 (true)	1 (true)	0 (false)

Підпрограми роботи з примітивними типами даних

function **CHR** (X : byte) : char;	System
Повертає символ з певним номером в ASCII таблиці.	
Функція Chr(X) повертає символ з порядковим номером X (в таблиці ASCII).	

procedure **DEC** (var X[; N : longint]);	System
Зменшує значення змінної.	
Параметр X – змінна порядкового типу або змінна типу PChar, якщо допускається розширений синтаксис; N – вираз цілого типу. Значення X зменшується на 1, якщо параметр N не визначено, або на N, якщо параметр N визначено, тобто Dec(X) відповідає X:=X–1, а Dec(X, N) відповідає X:=X–N. За допомогою Dec генерується більш оптимальний код, особливо корисний у щільному циклі.	

procedure **EXCLUDE** (var S : set of T; I : T);	System
Виключає елемент з множини.	
Параметр S – змінна типу Set, а I – вираз типу, сумісного з базовим типом для S. Елемент I виключається з множини S. Конструкція Exclude(S, I) відповідає виразу S:=S–[I], проте процедура Exclude генерує більш ефективний код.	

procedure **FILLCHAR** (var X; Count : word; Value);	System
Заповнює певну кількість (Count) безперервних байт певним значенням (типу Byte або Char).	
X – буфер, котрий потрібно заповнити; Count – кількість символів; Value – заповнювач (типу Byte або Char). Ця функція не виконує перевірки діапазону.	

function **HI** (X) : byte;	System
Повертає старший байт параметру.	
Параметр X – вираз типу Integer або Word. Функція High повертає старший байт X, знак не враховується.	

function **HIGH** (X) : <тип параметру>	System
Повертає максимальне значення в діапазоні параметра.	
Тип результату дорівнює типу параметру X або індексному типу параметра X, де X є ідентифікатором типу або посиланням на змінну.	

Для цього типу:	**High повертає**
Порядковий тип	Максимальне припустиме значення типу
Масив	Максимальне припустиме значення індексу масиву
Рядковий тип	Оголошену довжину рядка
Відкритий масив	Значення типу Word, що містить кількість елементів у відкритому масиві мінус один
Рядковий параметр	Значення типу Word, що містить кількість елементів у рядковому параметрі мінус один

procedure **INC** (var X [; N : longint]);	System
Збільшує значення змінної.	
Параметр X – змінна порядкового типу або змінна типу PChar, якщо допускається розширений синтаксис; N – вираз цілого типу. Значення X збільшується на 1, якщо параметр N не визначено, або на N, якщо параметр N визначено, тобто Inc(X) відповідає X:=X+1, а Inc(X, N) відповідає X:=X+N. За допомогою Inc генерується більш оптимальний код, особливо корисний у щільному циклі.	

procedure **INCLUDE** (var S : set of T; I : T);	System
Додає елемент до множини.	
Параметр S – змінна типу Set, а I – вираз типу, сумісного з базовим типом для S. Елемент I додається до множини S. Конструкція Include(S, I) відповідає виразу S:=S+[I], проте процедура Include генерує більш ефективний код.	

function **INT** (X : real) : real;	System
Повертає цілу частину параметра.	
Параметр X – вираз дійсного типу. Результат – ціла частина X, тобто значення X, що округлено до нуля.	

function **LO** (X) : byte;	System
Повертає молодший байт аргументу.	
Параметр X – вираз типу Integer або Word. Lo повертає молодший байт параметра X, знак не враховується.	

function **LOW** (X) : <тип параметра>	System
Повертає мінімальне значення в діапазоні параметра.	
Тип результату дорівнює типу параметра X або індексному типу параметра X, де X є ідентифікатором типу або посиланням на змінну.	

Для цього типу:	**Low повертає**
Порядковий тип	Мінімальне припустиме значення типу
Масив	Мінімальне припустиме значення індексу масиву
Рядковий тип	0

Відкритий масив	0
Рядковий параметр	0

procedure **MOVE** (var Source, Dest; Count : word);	System
Копіює байти з Source до Dest.	
Копіює певне число байт (Count) з Source до Dest. Ніяка перевірка діапазону не виконується. Завжди, коли це можливо, використовуйте SizeOf, щоб визначати кількість байт для копіювання.	

function **ORD** (X) : longint;	System
Повертає порядкове значення виразу порядкового типу.	
Параметр X – вираз порядкового типу. Результат має тип LongInt, а його значення – порядковий номер X в типі.	

function **PRED** (X) : <тип параметра>;	System
Повертає попередника параметра.	
Параметр X – вираз порядкового типу. Pred(X) поверне результат того самого типу, що і X, який є попередником X в цьому типі. Наприклад, Pred('B') = 'A'; Pred(9) = 8.	

function **SIZEOF** (T) : integer;	System
Повертає кількість байт, що займаються параметром.	
Параметр T – ідентифікатор типу або посилання на змінну. Повертає кількість байт, що необхідна для розміщення змінної певного типу даних в пам’яті. Коли використовується для об’єктних типів, котрі мають віртуальну таблицю методів (VMT), SizeOf повертає розмір, що збережено до VMT.	

function **SUCC** (X) : <тип параметра>;	System
Повертає послідовника параметра.	
Параметр X – вираз порядкового типу. Succ(X) поверне результат того самого типу, як X, що є послідовником X в цьому типі. Наприклад, Succ('B') = 'C'; Succ(9) = 10; Succ(FALSE) = TRUE.	

function **SWAP** (X) : <тип параметра>;	System
Переставляє старший та молодший байти параметра.	
X – вираз типу Integer або Word.	

function **UPCASE** (Ch : char) : char;	System
Перетворює символ з нижнього до верхнього регістру.	
Ch – вираз типу Char. Результат типу Char, що перетворено з нижнього до верхнього регістру. Символьні значення, що не знаходяться в діапазоні ['a' ..'z'], залишаються незмінними.	

Підпрограми обробки рядкового типу даних

function **CONCAT** (S1 [, S2,..., Sn] : string) : string;	System
Конкатенує декілька рядків.	
Кожний параметр – вираз з рядковим типом. Результат – рядок, що отриманий при конкатенуванні всіх рядкових параметрів. Якщо довжина отриманого рядка складає більш ніж 255 символів, то рядок скорочується після 255-го символу. Той самий результат можна отримати при використанні оператора додавання (+): S:='ABC'+'DEF'.	

function **COPY** (S : string; Index : integer; Count : integer) : string;	System
Повертає підрядок рядка.	
Параметр S – вираз з рядковим типом. Index та Count – вирази цілого типу. Функція Copy повертає підрядок рядка S, що містить Count символів, починаючи з символу з номером Index. Якщо значення Index більше, ніж довжина рядка S, то Copy повертає порожній рядок. Якщо значення Count більше, ніж кількість символів, що залишилися в рядку з позиції Index до кінця рядка, то повертається Length(S)-Index символів.	

procedure **DELETE** (var S : string; Index : integer; Count : integer);	System
Видаляє підрядок з рядку.	
Параметр S – змінна з рядковим типом. Index та Count – цілі вирази. Delete видаляє Count символів з рядка S, починаючи з позиції Index. Якщо значення Index більше, ніж довжина S, то ніякі символи не видаляються. Якщо Count визначає більшу кількість символів, ніж залишається в рядку з позиції Index до кінця рядка, то видаляється залишок рядка.	

procedure **INSERT** (Source : string; var S : string; Index : integer);	System
Вставляє підрядок в рядок.	
Параметр Source – вираз із рядковим типом. Параметр S – змінна з рядковим типом будь-якої довжини. Index – вираз цілого типу. Процедура Insert вставляє рядок Source в рядок S з позиції з номером Index. Якщо створений рядок довше за 255 символів, то він усікається після 255-го символу.	

function **LENGTH** (S : string) : integer;	System
Повертає динамічну довжину рядка.	
Працює аналогічно виразу Ord(S[0]).	

function **POS** (Substr : string; S : string) : byte;	System
Шукає підрядок в рядку.	
Параметри Substr та S – рядкові вирази. Pos шукає перше входження рядка Substr в рядок S та повертає ціле значення, котре є індексом першого символу Substr всередині S. Якщо рядок Substr не знайдено, то Pos повертає нуль.	

procedure **STR** (X [: width [: decimals]]; var S : string);	System
Перетворює число в рядок.	
Перетворює числове значення X в рядкове представлення цього числа, котре можна виводити такими операторами, як Write та OutText.	

procedure **VAL** (S; var V; var Code : integer);	System
Перетворює рядкове значення в його числове представлення.	
S – змінна з рядковим типом. Повинна представляти послідовність символів, що формують знакове ціле чи дійсне число. V – змінна цілого чи дійсного типу. Code – змінна типу Integer. Перетворює рядкове значення (S) в його числове представлення, як це трапляється при читанні з текстового файлу за допомогою Read. Code – позиція, в котрій виникла помилка при перетворенні, або нуль, якщо помилки не було.	

Підпрограми введення–виведення

procedure **BLOCKREAD** (var F : file; var Buf; Count : word [; var Result : word]);	System
Зчитує один і більше записів з файлу в змінну.	
F – нетипізована файлова змінна; Buf – будь-яка змінна; Count – вираз типу Word; Result – змінна типу Word. Процедура BlockRead зчитує Count чи меншу кількість записів з файлу F в блок пам'яті, що починається з першого байта, зайнятого змінною Buf. Реальна кількість зчитаних записів (≤Count) повертається в необов'язковому параметрі Result. Якщо параметр Result не визначено, то у випадку, коли кількість зчитаних записів не дорівнює параметру Count, трапляється помилка введення / виведення. Весь зчитаний блок займає максимум Count * RecSize байт, де RecSize – розмір запису, що визначається при відкриванні файлу (або 128 байт, якщо розмір запису не було визначено). Якщо Count * RecSize більше, ніж 64 Кб, то виникає помилка. Параметр Result є необов'язковим. Якщо весь блок було зчитано, то Result буде дорівнювати Count. Інакше, у випадку, якщо Result менший, ніж Count, кінець файлу буде досягнуто раніше, ніж буде завершене зчитування блоку. В такому випадку, якщо параметр RecSize був більше, ніж 1, Result поверне кількість цілком зчитаних записів. Вказівник поточної позиції файлу переміщується на кількість записів, що дорівнює значенню параметра Result. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Щоб використовувати цю процедуру, файл повинен бути відкритий для зчитування.	

procedure **BLOCKWRITE** (var F : file; var Buf; Count : word [; var Result : word]);	System
Записує одну чи більшу кількість записів зі змінної до файлу.	
F – нетипізована файлова змінна; Buf – будь-яка змінна; Count – вираз типу Word; Result – змінна типу Word. Процедура BlockWrite записує Count або меншу кількість записів до файлу F з блоку пам'яті, що починається з першого байта, який зайнято змінною Buf. Реальна кількість збережених записів (≤Count) повертається в необов'язковому параметрі Result. Якщо параметр Result не визначено, то у випадку, коли кількість записаної інформації не дорівнює параметру Count, трапляється помилка введення / виведення. Увесь записаний блок займає максимум Count * RecSize байт, де RecSize – розмір запису, що визначається при відкриванні файлу (або 128 байт, якщо розмір запису не було визначено). Якщо Count * RecSize більше, ніж 64 Кб, то виникає помилка. Параметр Result є необов'язковим. Якщо весь блок було записано, то Result буде дорівнювати Count. Інакше, у випадку, якщо Result менше, ніж Count, логічний диск заповниться раніше, ніж буде завершений запис блоку. В такому випадку, якщо параметр RecSize був більше, ніж 1, то Result поверне кількість цілком записаних записів. Вказівник поточної позиції файлу переміщується на кількість записів, що дорівнює значенню параметра Result. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Щоб використовувати цю процедуру, файл повинен бути відкритий для запису.	

function **IORESULT** : integer;	System
Повертає статус останньої виконаної операції введення / виведення.	
Якщо помилки не було, повертається нуль. Щоб контролювати помилки введення / виведення за допомогою функції IOResult, опція перевірки введення / виведення має бути вимкнена ({$I-}). Якщо трапляється помилка введення / виведення та перевірку введення / виведення вимкнено, то всі наступні операції введення / виведення ігноруються, доки не буде зроблено звернення до функції IOResult, за якого очиститься внутрішній стан помилки.	

procedure **READ** (F , V1 [, V2,...,Vn]); (типізовані файли) procedure **READ** ([var F : text;] V1 [, V2,...,Vn]); (текстові файли)	System
Для типізованих файлів зчитує компонент файлу в змінну. Для текстових файлів зчитує одну чи більшу кількість значень в одну чи більшу кількість змінних.	

Для рядкових змінних Read зчитує всі символи до (але не включно) наступного маркера кінця рядка або доки Eof(F) не стане дорівнювати True. Read не переходить до наступного рядка після зчитування. Якщо отриманий в результаті рядок довше, ніж максимальна довжина рядкової змінної, то вона усікається. Після першого Read всі наступні виклики Read будуть бачити маркер кінця рядка та повертати рядок нульової довжини. Використовуйте декілька звернень до ReadLn, щоб зчитати декілька рядкових значень. Коли ввімкнено опцію Extended Syntax, процедура Read може зчитувати рядки з нульовим закінченням до нуль-заснованих масивів символів.

Для змінних цілих та дійсних типів Read буде пропускати будь-які пробіли, мітки табуляції або маркери кінця рядка, що передували числовому рядку. Якщо числовий рядок не відповідає формату, що очікується, трапляється помилка введення / виведення, інакше змінній присвоюється отримане значення. Наступний Read почнеться з пробілу, знаку табуляції або маркера кінця рядка, котрі завершували числовий рядок.

procedure **READLN** ([var F : text;] V1 [, V2, ...,Vn]);	System
Виконується процедура Read, потім виконується перехід на наступний рядок файлу.	
Після виконання Read, ReadLn переходить на початок наступного рядка файлу.	

function **SEEKEOF** [(var F: text)]: boolean;	System
Повертає стан кінця файлу.	
Використання можливе тільки на текстових файлах. Файл повинен бути відкритий. Повертає True у випадку, коли вказівник позиції файлу знаходиться в кінці файлу або від поточної позиції вказівника файлу до кінця файлу містяться тільки пробіли, знаки табуляції та переходи до наступного рядка.	

function **SEEKEOLN** [(var F: text)]: boolean;	System
Повертає стан кінця рядка в файлі.	
Використання можливе тільки на текстових файлах. Файл повинен бути відкритий. Повертає True у випадку, коли вказівник позиції файлу знаходиться в кінці рядка або від поточної позиції вказівника файлу до кінця рядка містяться тільки пробіли та знаки табуляції.	

procedure **SETTEXTBUF** (var F : text; var Buf [; Size : word]);	System
Призначає буфер введення / виведення на текстовий файл.	

Процедуру SetTextBuf не можна використовувати для відкритого файлу, проте її можна викликати відразу після виконання Reset, ReWrite та Append. Якщо Ви викликаєте SetTextBuf для відкритого файлу під час операцій введення / виведення, то це може викликати втрату даних через зміну буфера. Borland Pascal не гарантує, що буфер буде існувати під час всіх операцій введення / виведення до файлу. Поширена помилка полягає в тому, що використовують локальну змінну як буфер, а потім використовують файл поза процедурою, в котрій було визначено буфер.

procedure **WRITE** (F, V1 [, V2,...,Vn]); (типізовані файли) procedure **WRITE** ([var F : text;] P1 [,P2,...,Pn]); (текстові файли)	System
Для типізованих файлів записує змінну до компоненти файлу. Для текстових файлів записує одну чи більшу кількість змінних до файлу.	
Файл повинен бути відкритим для запису. Текстові файли Кожний параметр P – параметр запису, який вміщує вираз, значення котрого має бути записано до файлу. Параметр запису може також містити специфікатори ширини поля та кількості знаків після десятинної крапки. Кожен вираз виведення повинен мати тип Char, Integer, Real, String, Packed String або Boolean. Типізовані файли Кожний параметр V – змінна того самого типу, що й тип компонентів у файлі F. Після запису кожної змінної поточний вказівник файлу переміщується до наступної компоненти. Якщо поточний вказівник файлу знаходиться в кінці файлу, то файл розширюється. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **WRITELN** ([var F : text;] P1 [, P2, ...,Pn]);	System
Виконує процедуру Write, потім записує маркер кінця рядка в файл.	
Процедура WriteLn є розширенням процедури Write, оскільки вона визначена тільки для текстових файлів. Після виклику процедури Write, WriteLn записує маркер кінця рядка (CR/LF) в файл. Звернення типу WriteLn(F) записує маркер кінця рядка до файлу F. Виклик WriteLn без параметрів відповідає виклику WriteLn(Output).	

Підпрограми керування процесом виконання програми та роботи з оточенням програми

procedure **EXIT;**	System
Негайно виходить з поточного блоку програми.	
Якщо поточний блок – основна програма, то виклик Exit завершить виконання програми.	

procedure **HALT** [(Exitcode : word)];	System
Зупиняє виконання програми та виходить в операційну систему.	
Викликає виконання всіх процедур виходу. Код завершення може бути розглянутий батьківським процесом з використанням змінної DosExitCode в модулі DOS, або через перевірку змінної ERRORLEVEL в *.BAT файлі DOS.	

function **PARAMCOUNT** : word;	System
Повертає кількість параметрів, що передано в програму через командний рядок.	
Пробіли та мітки табуляції використовуються як роздільники.	

function **PARAMSTR** (Index : word) : string;	System
Повертає певний параметр командного рядка.	
ParamStr повертає параметр з номером Index з командного рядка чи порожній рядок, якщо Index більше, ніж ParamCount. ParamStr(0) повертає шлях та ім’я програми, що виконується (наприклад, C:\BP\MYPROG.EXE).	

procedure **RUNERROR** [(Errorcode : byte)];	System
Зупиняє виконання програми.	
Генерує помилку під час виконання програми із заданим номером в поточному операторі.	

Математичні підпрограми

function **ABS** (X) : (тип параметру);	System
Повертає абсолютне значення параметра.	
Параметр X – вираз дійсного або цілого типу. Результат того ж самого типу, що й X. Є абсолютним значенням X.	

function **ARCTAN** (X : real) : real;	System
Повертає арктангенс параметра.	
У мові Turbo Pascal немає вбудованої функції Tan, проте тангенс може бути обчислено за допомогою виразу: Sin(X) / Cos(X)	

function **COS** (X : real) : real;	System
Повертає косинус параметра (X – кут в радіанах).	
Параметр X – вираз дійсного типу. Результат – косинус числа X, де X – кут в радіанах.	

function **EXP** (X : real) : real;	System
Повертає експоненту параметра.	
Значення *e* зведене в ступінь X, де *e* – підстава натуральних логарифмів.	

function **FRAC** (X : real) : real;	System
Повертає дробову частину аргументу.	
Параметр X – вираз дійсного типу. Результат – дробова частина X, тобто Frac(X) = X–Int(X).	

function **LN** (X : real) : real;	System
Повертає натуральний логарифм аргументу.	
Повертає натуральний логарифм дійсного виразу X.	

function **ODD** (X : longint) : boolean;	System
Перевіряє параметр на непарність.	
Значення функції Odd(X) дорівнює True, якщо X – непарне число.	

function **PI** : real;	System
Повертає значення Pi, що дорівнює 3.1415926535897932385.	
Точність може змінюватися залежно від того, чи знаходиться компілятор в режимі 80x87 або тільки в режимі програмної емуляції сопроцесора.	

function **RANDOM** [(Range : word)] : <тип параметру>;	System
Повертає випадкове число.	
Якщо параметр Range не визначено, то результатом буде випадкове число типу Real в діапазоні $0 \le X < 1$. Якщо параметр Range визначено, то результатом буде випадкове число в діапазоні $0 \le X <$ Range. Якщо Range дорівнює 0, то буде повернене значення 0. Щоб ініціалізувати генератор випадкових чисел, треба викликати процедуру Randomize або присвоїти значення змінної RandSeed.	

procedure **RANDOMIZE;**	System
Ініціалізує вбудований генератор випадкових чисел довільним значенням (що отримано із системного годинника).	

function **ROUND** (X : real) : longint;	System
Округлює значення дійсного типу до значення цілого типу.	
X – вираз із дійсним типом. Round повертає значення типу Longint, котре є значенням X, що округлено до найближчого цілого числа. Якщо X знаходиться рівно посередині між двома цілими числами, то результатом буде число з найбільшою абсолютною величиною. Якщо округлене значення X не знаходиться всередині допустимого діапазону Longint, то трапляється помилка часу виконання програми.	

function **SIN** (X : real) : real;	System
Повертає синус параметра.	
X – вираз дійсного типу. Повертає синус кута X в радіанах.	

function **SQR** (X): (тип параметра);	System
Повертає квадрат параметра.	
Число X – або дійсний, або цілий вираз. Результат того самого типу, що й X, є квадратом числа X.	

function **SQRT** (X : real) : real;	System
Повертає квадратний корінь аргумента.	
X – вираз дійсного типу. Результатом буде квадратний корінь з X.	

function **TRUNC** (X : real) : longint;	System
Усікає значення дійсного типу до значення цілого типу.	
X – вираз дійсного типу. Trunc повертає значення Longint, котре є значенням X, усіченим до нуля. Якщо усічене значення X не знаходиться всередині допустимого діапазону Longint, то трапляється помилка часу виконання програми.	

Підпрограми роботи з файлами

procedure **APPEND** (var F : text);	System
Відкриває існуючий файл для продовження запису в файл.	
Параметр F – змінна текстового файлу, що має бути зв'язана із зовнішнім файлом за допомогою виклику процедури Assign. Append відчиняє існуючий зовнішній файл з іменем, визначеним в файловій змінній F. Якщо зовнішнього файлу з таким іменем не існує, то трапляється помилка введення / виведення. Якщо F вже відкритий, то він закривається та знову відкривається. Поточна позиція файлу встановлюється на кінець файлу. Якщо в останньому 128-байтовому блоці файлу існує символ Ctrl+Z (символ, з кодом 26), то поточна позиція файлу встановлюється так, щоб перезаписати перший зустрічний Ctrl+Z, в блоці. Отже, до файлу, котрий завершується символом Ctrl+Z, може бути додано текст. Після звернення до Append, F стає файлом тільки для запису, а вказівник позиції файлу встановлюється на його кінець. У режимі {$I–} функція IOResult поверне нуль, якщо операція була успішна, інакше вона поверне ненульовий від нуля код помилки.	

procedure **ASSIGN** (var F; NAME : string);	System
Призначає файловій змінній ім'я зовнішнього файлу.	
F – файлова змінна будь-якого файлового типу; Name – вираз рядкового типу або вираз типу PChar, якщо припускається розширений синтаксис. Всі подальші операції зі змінною F насправді виконуються із зовнішнім файлом з іменем Name. Після звернення до Assign, зв'язок між змінною F та зовнішнім файлом продовжує існувати, доки для змінної F не буде зроблено ще одне перепризначення.	

Ім’я файлу складається зі шляху: нульової чи більшої кількості імен каталогів, розділених символом «\», що супроводжується іменем файлу: Drive:\ DirName\...\DirName\FileName.Ext.

Якщо шлях починається із символу «\», то він знаходиться в кореневому каталозі, в іншому випадку – він знаходиться в поточному каталозі. Drive – ідентифікатор дисковода (A-Z). Якщо Drive та двокрапка відсутні, то використовується дисковод за замовчуванням. Ім’я файлу має ті ж самі обмеження, що прийнято для імен файлів операційної системи DOS. Максимальна довжина всього ім’я файлу разом зі шляхом – 79 символів.

Якщо Name – порожній рядок, то виникає спеціальний випадок, та змінна F є зв’язаною зі стандартним файлом введення або виведення.

Функція Assign не може використовуватися для вже відкритих файлів.

procedure **CLOSE** (var F);	System

Закриває раніше відкритий файл.

Параметр F – файлова змінна будь-якого типу, що зв’язана з попередньо відкритими процедурами Reset, ReWrite або Append файлом. Зовнішній файл, що зв’язано з F, повністю обробляється, а потім закривається, при цьому звільняється дескриптор файлу DOS для подальших звернень до нього.

У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.

function **EOF** (var F) : boolean;	System

Повертає стан кінця файлу.

Параметр F (якщо визначено) є змінною будь-якого файлового типу. Якщо параметр F відсутній, то розуміється стандартна файлова змінна Input. Eof(F) повертає значення True, якщо вказівник поточної позиції файлу знаходиться поза останнього символу файлу або якщо файл взагалі не містить компонентів, інакше Eof(F) повертає False.

У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.

function **EOLN** [(var F : text)] : boolean;	System

Повертає стан кінця рядка текстового файлу.

Параметр F (якщо визначено) є змінною текстового файлу. Якщо параметр F відсутний, то розуміється стандартна файлова змінна Input. Eoln(F) повертає значення True, якщо вказівник поточної позиції файлу знаходиться на маркері кінця рядка або якщо Eof(F) дорівнює True, інакше Eoln(F) повертає False.

У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.

procedure **ERASE** (var F);	System
Стирає зовнішній файл з диску.	
Параметр F – файлова змінна будь-якого файлового типу. Зовнішній файл, що зв'язано зі змінною F, видаляється. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Використання Erase на відкритому файлі неприпустиме.	

function **FILEPOS** (var F) : longint;	System
Повертає поточну позицію вказівника файлу.	
Параметр F – файлова змінна. Якщо вказівник поточної позиції файлу знаходиться на початку файлу, то FilePos(F) поверне нуль. Якщо вказівник поточної позиції файлу знаходиться в кінці файлу, тобто, якщо Eof(F) = True, то значення FilePos(F) дорівнює значенню FileSize(F). У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Функція не може використовуватися для текстових файлів. Файл повинен бути відкритим.	

function **FILESIZE** (var F) : longint;	System
Повертає поточний розмір файлу.	
Параметр F – файлова змінна. FileSize(F) повертає кількість компонентів в F. Якщо файл порожній, то FileSize(F) повертає нуль. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Функція не може використовуватися для текстових файлів. Файл повинен бути відкритим.	

procedure **FLUSH** (var F : text);	System
Очищує буфер текстового файлу, відкритого для виведення.	
Параметр F – змінна текстового файлу. Якщо текстовий файл було відкрито для виведення з використанням процедур ReWrite або Append, то виклик Flush очистить буфер файлу. Це гарантує, що всі символи, що записані в цей час в файл, будуть записані на диск. Виклик Flush не має ніякого ефекту для файлів, відкритих на введення. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **RENAME** (var F; NewName);	System
Змінює ім'я зовнішнього файлу.	
Параметр F – змінна будь-якого файлового типу. Параметр NewName – типу String або PChar, якщо ввімкнено розширений синтаксис. Зовнішній файл, пов'язаний зі змінною F, перейменовується на NewName. Подальші операції на F виконуються вже із зовнішнім файлом з новим іменем. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **RESET** (var F [: file; RecSize : word]);	System
Відкриває існуючий файл.	
Параметр F – змінна будь-якого файлового типу, що пов'язана із зовнішнім файлом, з використанням процедури Assign. Параметр RecSize – необов'язковий параметр, котрий може бути визначений тільки якщо F – нетипізований файл. В цьому випадку RecSize визначає розмір блоку, що потрібно використовувати при передаванні даних. Якщо RecSize відсутній, приймається заданий за замовчуванням розмір блоку в 128 байт. Reset відкриває існуючий зовнішній файл з іменем, призначеним в змінній F. Трапляється помилка, якщо зовнішній файл із заданим іменем не існує. Якщо файл F вже відкрито, то він спочатку закривається, а потім знову відкривається. Поточна позиція вказівника встановлюється на початок файлу. Якщо F – текстовий файл, то він відкривається тільки для читання. Після звернення до Reset, Eof(F) = True, якщо файл порожній, інакше Eof (F) = False. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **REWRITE** (var F : file [; RecSize : word]);	System
Створює та відкриває новий файл.	
Параметр F – змінна будь-якого файлового типу, що пов'язана із зовнішнім файлом, з використанням процедури Assign. RecSize – необов'язковий параметр, що визначає розмір блоку, котрий треба використовувати при передаванні даних. Якщо RecSize відсутній, приймається заданий за замовчуванням розмір блоку в 128 байт. Процедура ReWrite створює новий зовнішній файл з іменем, що призначено в змінній F. Якщо зовнішній файл з тим самим іменем вже існує, то він видаляється, а на його місці створюється новий порожній файл. Якщо F вже відкрито, то він спочатку закривається, а потім знову створюється. Поточна позиція вказівника встановлюється на початок порожнього файлу. Якщо F – текстовий файл, то F відкривається тільки для запису. Після звернення до ReWrite, значення Eof(F) дорівнює True. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **SEEK** (var F; N : longint);	System
Переміщує поточний вказівник позиції файлу на певний компонент.	
F – змінна будь-якого файлового типу, крім текстового. N – вираз типу Longint. Вказівник позиції файлу F зсувається на номер компонента N. Номер першого компонента файлу дорівнює нулю. Щоб розширити файл, можна зсунути вказівник на один компонент поза останній компонент в файлі. Тобто, оператор Seek(F, FileSize(F)) зсуває поточний вказівник позиції файлу на кінець файлу.	

У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Не може бути використана на текстових файлах. Файл повинен бути відкритим.

procedure **TRUNCATE** (var F);	System
Усікає файл в поточній позиції файлу.	
F – файлова змінна будь-якого типу, крім текстового. Все, що знаходиться поза поточною позицією файлу, видаляється, та повертається ознака кінця файлу (EOF(F) = True). У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. Не може бути використана на текстових файлах. Файл повинен бути відкритим.	

Підпрограми роботи з файловою системою

procedure **CHDIR** (S : string);	System
Змінює поточний каталог.	
Поточний каталог змінюється на каталог, що визначається параметром S. Якщо в S міститься вказівник на інший логічний диск, то поточний диск також змінюється. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

procedure **GETDIR** (D : byte; var S : string);	System
Повертає поточний каталог заданого диска.	
Параметр D: 0 – поточний диск, 1 – диск A, 2 – диск B, 3 – диск C і так далі... Не виконує перевірку помилок. Якщо диск, що задано параметром D неприпустимий, то в рядку S повертається X:\, ніби це кореневий каталог неприпустимого диска. Процедура GetCurDir виконує ту саму функцію, що й GetDir, але як один із параметрів використовується рядок з нульовим закінченням, замість рядка стилю Pascal.	

procedure **MKDIR** (S : string);	System
Створює підкаталог.	
Створює новий підкаталог з іменем, що визначено рядком S. Остання частина рядка не може бути іменем існуючого файлу. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки. CreateDir виконує ту саму функцію, що й MkDir, але як параметр використовується рядок з нульовим закінченням.	

procedure **RMDIR** (S : string);	System
Видаляє порожній каталог.	
Видаляє підкаталог зі шляхом, що зазначено в S. Якщо шляху не існує, каталог не є порожнім, або якщо це поточний каталог, то виникає помилка введення / виведення. У режимі {$I-} функція IOResult поверне нуль, якщо операція була успішною, інакше вона поверне відмінний від нуля код помилки.	

Підпрограми роботи з динамічною пам'яттю

function **ADDR** (X) : pointer;	System
Повертає адресу визначеного об'єкта.	
Параметр X – ідентифікатор змінної, процедури чи функції. Результат – вказівник на X. Як й NIL, результат функції Addr сумісний з усіма типами вказівників.	

function **ASSIGNED** (var P) : boolean;	System
Визначає, чи дорівнює вказівник або процедурна змінна нулю.	
Параметр P повинен бути вказівником або змінною процедурного типу. Assigned(P) відповідає виразу P<>Nil для вказівника та @P<>Nil – для процедурної змінної. Повертає True, якщо P дорівнює Nil, інакше повертає значення False.	

procedure **DISPOSE** (var P : pointer [, Destructor]);	System
Звільняє місце, що займається динамічною змінною в пам'яті.	
Після звернення до Dispose, значення вказівника P стає невизначеним, та посилатися на нього є помилкою. Не повинна використовуватися разом з процедурами Mark та Release. Процедура Dispose може також звільняти пам'ять, що займається об'єктом, який розміщується в динамічній пам'яті, якщо вказується деструктор цього об'єкта як другий параметр. Якщо P не вказує на область пам'яті в динамічній пам'яті, то виникає помилка часу виконання програми.	

procedure **FREEMEM** (var P : pointer; Size : word);	System
Звільняє пам'ять, що займається динамічною змінною зазначеного розміру.	
Параметр P – змінна будь-якого вказівникового типу, попередньо розміщена в пам'яті процедурою GetMem або та, котрій було присвоєне значення оператором присвоєння. Параметр Size – вираз, що визначає розмір динамічної змінної в байтах, пам'ять, що потрібно звільнити. Він повинен дорівнювати кількості байт, що попередньо розміщено для цієї змінної процедурою GetMem.	

Процедура FreeMem знищує змінну, що пов'язану з P, та звільняє пам'ять, що займається цією змінною. Якщо P не вказує на область пам'яті в динамічній пам'яті, то виникає помилка часу виконання програми. Після звернення до FreeMem, значення P стає невизначеним, та при спробі звернення до P^ виникає помилка.

Процедуру не можна використовувати разом з Mark або Release.

procedure **GETMEM** (var P : pointer; Size : word);	System

Створює динамічну змінну певного розміру та записує її адресу в пам'яті в задану змінну.

Параметр P – змінна будь-якого вказівникового типу. Параметр Size – вираз, що визначає розмір динамічної змінної в байтах. Нещодавно створена змінна може бути викликана як P^.

Якщо в динамічній пам'яті недостатньо вільного місця, виникає помилка періоду виконання програми.

Найбільший блок, що може бути безпечно розміщений в динамічній пам'яті, дорівнює 65,528 байт (64K-$8).

procedure **NEW** (var P : pointer [, Init : constructor]);	System

Створює нову динамічну змінну та встановлює на неї вказівник.

Процедура New може також ініціалізувати об'єкт, розміщений в динамічній пам'яті, якщо вказується конструктор цього об'єкта як другий параметр.

function **NEW** (T [, Init : constructor]) : pointer;	System

Створює нову динамічну змінну та встановлює на неї вказівник.

Параметр, що передається до New, – тип вказівника на об'єкт, а не власне вказівник. Ця функціональна форма New може бути використана для всіх типів даних, з об'єктними включно.

function **MAXAVAIL** : longint;	System

Повертає розмір найбільшого безперервного вільного блоку в динамічній пам'яті.

Повертає більший з:

– найбільших вільних блоків всередині області менеджера керування динамічної пам'яті;

– глобальної динамічної пам'яті Windows.

Значення відповідає розміру найбільшої динамічної змінної, яка може бути розміщена.

function **MEMAVAIL** : longint;	System

Повертає розмір всієї вільної пам'яті в динамічній пам'яті.

MemAvail повертає суму розмірів всіх вільних блоків в динамічній пам'яті. Безперервний блок пам'яті розміру, що повертається, навряд чи буде доступний через фрагментацію динамічної пам'яті.

У реальному режимі DOS MemAvail вираховується додаванням розмірів всіх вільних блоків нижче вказівника динамічної пам’яті до розміру вільної пам’яті вище вказівника динамічної пам’яті. Програма може визначати мінімальні та максимальні вимоги до динамічної пам’яті, використовуючи директиву {$M}.

У захищеному режимі DOS і Windows MemAvail вираховує кількість вільної пам’яті, викликаючи функцію GetFreeSpace та додаючи до цього розмір кожного вільного блоку в області підрозділів програми керування динамічною пам’яттю.

function **OFS** (X) : word;	System
Повертає зміщення певного об’єкта.	
Параметр X – це будь-яка змінна або ідентифікатор процедури або функції. Результат типу Word – зміщення адреси X в пам’яті.	

function **PTR** (Seg, Ofs : word) : pointer;	System
Перетворює адресу у сегмент : зміщення на вказівник.	
Seg та Ofs – вирази типу Word. Результат – вказівник адреси Seg:Ofs. Результат, що повертається Ptr, сумісний з усіма типами вказівників (як і NIL). Наприклад: If Byte(Ptr($40, $49)^)=7 Then WriteLn('Відеорежим – монохромний');	

function **SEG** (X) : word;	System
Повертає сегмент певного об’єкта.	
Параметр X – будь-яка змінна або ідентифікатор процедури або функції. Результат типу Word, є сегментною частиною адреси X.	

Помилки періоду виконання програми

№	Назва	Опис
1	Invalid function number / Неправильний номер функції	Було виконано виклик неіснуючої функції DOS.
2	File not found / Файл не знайдено	Підпрограми **Reset, Append, Rename** та **Erase** сповіщають про цю помилку, якщо ім'я, що призначене файловій змінній, не визначає існуючий файл на диску.
3	Path not found / Шлях не знайдено	Підпрограми **Reset, Rewrite, Append, Rename** та **Erase** сповіщають про цю помилку, якщо ім'я, що призначене файловій змінній, недопустиме або визначає неіснуючий підкаталог. Підпрограми **ChDir, MkDir** або **RmDir** сповіщають про цю помилку, якщо шлях неприпустимий або якщо він визначає неіснуючий підкаталог.
4	Too many open files / Забагато відкритих файлів	Підпрограмі **Reset, Rewrite** та **Append** сповіщають про цю помилку, якщо програма відкрила забагато файлів. За замовчуванням DOS ніколи не дозволяє відкрити більше 15 файлів на процес. Якщо Ви отримуєте цю помилку, але впевнені, що відкритих файлів менше за 15, то це означає, що у файлі CONFIG.SYS немає рядка FILES = XX (або xx має замале значення). Збільшить номер xx до певного прийнятного значення, наприклад до 20.
5	File access denied / Доступ до файлу закрито	**Reset** чи **Append** – якщо FileMode дозволяє запис, проте призначене файловій змінній ім'я визначає каталог чи файл тільки для читання. **Rewrite** – в каталозі немає місця, призначене файловій змінній ім'я визначає каталог або існуючий файл тільки для читання. **Rename** – ім'я, що призначене файловій змінній, визначає каталог або нове ім'я визначає вже існуючий файл. **Erase** – ім'я, що призначене файловій змінній, визначає каталог або файл тільки для читання. **MkDir** – файл з таким іменем вже існує, в каталозі немає вільного місця або шлях визначає пристрій. **RmDir** – каталог непорожній, шлях не визначає каталог або шлях визначає кореневий каталог.

№	Назва	Опис
		Read або **BlockRead** (на типізованому чи нетипізованому файлі): файл не відкрито для читання. **Write** або **BlockWrite** (на типізованому чи нетипізованому файлі): файл не відкрито для запису.
12	Invalid file access code / Неправильний параметр доступу до файлу	Підпрограми **Reset** та **Append** сповіщають про цю помилку (на типізованих чи нетипізованих файлах), якщо значення **FileMode** неприпустиме.
15	Invalid drive number / Неправильний номер диску	**GetDir** сповіщає про цю помилку, якщо номер диска неприпустимий.
16	Cannot remove current directory / Не можу видалити поточний каталог	**RmDir** сповіщає про цю помилку, якщо шлях визначає поточний каталог.
17	Cannot rename across drives / Не можу змінювати ім’я між дисками	**Rename** сповіщає про цю помилку, якщо обидва файли знаходяться на різних дисках.
18	No more files / Немає більше файлів	Сповіщається в змінній **DosError,** коли при зверненні до **FindFirst** або **FindNext** на диску немає більше прийнятних файлів.
100	Disk read error / Дискова помилка читання	Процедура **Read** сповіщає про цю помилку на типізованому файлі, якщо є змога зчитувати будь-що після закінчення файлу.
101	Disk write error / Помилка записування на диск	Підпрограми **Close, Write, WriteLn** та **Flush** сповіщають про цю помилку, якщо на диску недостатньо вільного місця.
102	File not assigned / Файл не призначено	Підпрограми **Reset, Rewrite, Append, Rename** та **Erase** сповіщають про цю помилку, якщо файловій змінній не призначено ім’я за зверненням до **Assign.**
103	File not open / Файл не відкрито	Такі підпрограми сповіщають про цю помилку якщо файл не відкрито: **BlockRead, BlockWrite, Close, Eof, FilePos, FileSize, Flush, Read, Seek, Write.**
104	File not open for input / Файл не відкрито для введення	Такі підпрограми сповіщають про цю помилку, якщо текстовий файл не відкрито для введення: **Eof, Eoln, Read, ReadLn, SeekEof, SeekEoln.**
105	File not open for output / Файл не відкрито для виведення	Підпрограми **Write** та **WriteLn** сповіщають про цю помилку, якщо текстовий файл не відкрито для запису.
106	Invalid numeric format / Неприпустимий числовий формат	Підпрограми **Read** та **ReadLn** сповіщають про цю помилку, якщо числове значення, зчитане з текстового файлу, не відповідає потрібному числовому формату.

№	Назва	Опис
150	Disk is write-protected / Диск захищено від запису	Відбулася спроба запису на захищений від запису диск.
160	Device write fault / Помилка записування на пристрої	DOS зіткнулася з апаратною помилкою при спробі записати дані на пристрій.
161	Device read fault / Помилка читання з пристрою	DOS зіткнулася з апаратною помилкою при спробі зчитати дані з пристрою.
162	Hardware failure / Апаратний збій	DOS зіткнулася з апаратною відмовою на пристрої введення / виведення. Ця помилка може бути викликана спільним використанням або іншими мережевими помилками.
200	Division by zero / Ділення на нуль	Програма спробувала ділити на нуль в операціях «/», «mod» або «div».
201	Range check error / Помилка діапазону	Ця помилка сповіщається операторами, що компілюються в режимі {$R+}, під час виникнення таких ситуацій: – індекс масиву був поза діапазоном; – програма спробувала надати змінній значення поза діапазоном; – програма спробувала передати процедурі або функції значення поза діапазоном.
202	Stack overflow error / Помилка переповнення стека	Програма сповіщає про цю помилку при вході в процедуру чи функцію, що компілюється в режимі {$S+}, коли немає достатньо місця в стеку для розподілення локальних змінних підпрограми. Розмір стека встановлюється за допомогою директиви компілятора $M. Помилка переповнення стека може бути також викликана нескінченною рекурсією або підпрограмою на асемблері, котра некоректно працює зі стеком.
203	Heap overflow error / Помилка переповнення кучі	Підпрограми **New** та **GetMem** сповіщають про цю помилку, коли недостатньо вільного місця в кучі (Heap), для розподілення блоку потрібного розміру. Якщо ця помилка виникає під час виконання Вашої програми всередині середовища, спробуйте вийти з Turbo Pascal (File ⎪ Exit) та запустити програму з командного рядка DOS.
204	Invalid pointer operation / Неприпустима операція із вказівником	Підпрограми **Dispose** та **FreeMem** сповіщають про цю помилку, якщо вказівник дорівнює NIL або вказує на адресу поза кучею.
205	Floating point overflow / Переповнення операції з рухомою крапкою	Операція з рухомою крапкою повернула занадто велике число для Turbo Pascal (або для сопроцесора, якщо він є).

№	Назва	Опис
207	Invalid floating point operation / Неприпустима операція з рухомою крапкою	Виникла одна з наступних помилок з рухомою крапкою: – реальне значення, що передане до **Trunc** або **Round,** не може бути перетворене до Integer всередині LongInt діапазону; – параметр, переданий до функції **Sqrt,** був негативним; – параметр, переданий до функції **Ln,** дорівнює нулю або негативний; – виникло переповнення стека 80x87.
215	Arithmetic overflow error / Помилка арифметичного переповнення	Помилка сповіщається операторами, що компілюються в режимі {$Q+}, коли цілочисельна арифметична операція викликала переповнення або результат був поза діапазоном.
216	General Protection fault / Помилка захисту (GPF)	Ця помилка виникає, якщо відбувається спроба звернутися до пам’яті, доступ до котрої закрито для програми. Операційна система зупиняє програму та сповіщає, що виникла помилка захисту. Такі дії зазвичай викликають GPF: – завантаження констант в сегментні регістри; – виконання арифметичних операцій на сегментних регістрах селекторів; – використання сегментних регістрів задля тимчасового збереження; – запис до сегмента коду; – доступ до пам’яті поза локального адресного простору, що надано програмі; – перейменування нульових вказівників.

Розділ 3. Шаблони програмування

У цьому розділі буде надано деякі шаблони для організації інтерфейсу користувача та обробки даних. У формальних описах використовуються такі позначення:

<Блок> – логічне позначення деякого блоку;

[Блок] – блок, котрий може бути відсутнім;

жирний – зарезервовані слова;

… – повторення попередньої конструкції.

1. Шаблон організації програми

Програма мовою Pascal має таку структуру:

Формальний опис	Приклад

[<Директиви компілятора>]
 	
[**program** <Назва програми>;]
 	
[**uses** <Назва модуля>[, …]]
 	
[<Розділ опису констант>]
 	
[<Розділ опису типів даних користувача>]
 	
[<Розділ опису змінних>]
 	
[<Розділ опису міток>]
 	
[<Розділ опису підпрограм>]
 	
begin
  <Основна програма>
 	
end.
	{$N+}
	
	program Summ;
	
	uses crt, prn;
	
	const
	  N = 10;
	
	type
	  Tarr = **array** [1..N] **of** byte;
	
	var
	  a : Tarr;
	  i, s : integer;
	
	label
	  l1, l2;
	
	procedure add(**var** a: integer; b: integer)
	begin
	  a:=a + b;
	end;
	
	begin
	  for i:=1 **to** N **do**
	  add(s, a[i]);
	end.

<Директиви компілятора> – написані через пробіл глобальні та, можливо, локальні директиви компілятора.

<Розділ опису констант> – розділ, в якому йменуються всі глобальні константи. Має такий вигляд:

Формальний опис	Приклад
const <Ім'я константи> = <значення константи>; […]	const N = 10; Pi = 3.14; C = $1F;

<Розділ опису типів даних користувача> – розділ, в якому визначаються всі типи даних користувача. Має такий вигляд:

Формальний опис	Приклад
type <Ім'я типу даних> = <Опис типу даних>; […]	type Tarr = array [1..N] of byte; max255 = byte;

<Розділ опису змінних> – розділ, в якому описуються всі глобальні змінні. Має такий вигляд:

Формальний опис	Приклад
var <Ім'я змінної>[, …] : <Ім'я типу даних>; […]	var a : Tarr; i, s : integer;

<Розділ опису міток> – розділ, в якому описуються всі мітки переходу в програмі. Згідно з принципами структурного програмування, використання міток в програмі недопустиме, та конструкція залишена для сумісності з ранніми версіями мови Pascal. Розділ має такий вигляд:

Формальний опис	Приклад
label <Ім'я мітки> [, …];	label l1, l2;

<Розділ опису підпрограм> – розділ, в якому описуються всі підпрограми. Структура опису підпрограм буде наведена нижче.

<Основна програма> – розділ, в якому реалізується основний алгоритм програми. Основна програма повинна мати максимально просту та прозору структуру. Структура залежить від типу алгоритму.

1. Обчислювальний алгоритм

Обчислювальний алгоритм характеризується лінійною структурою та складається з чотирьох блоків:

Формальний опис	Приклад

<Блок ініціалізації змінних>
 	
<Блок введення даних>
 	
<Блок обчислень>
 	
<Блок виведення даних>
	s:=0;
	
	writeln('Введіть елементи масиву:');
	for i:=1 to N do begin
	  write('a[',i,'] = ');
	  readln(a[i]);
	end;
	
	for i:=1 to N do
	  s:=s+a[i];
	
	writeln('Сума елементів дорівнює ',S);

2. Циклічний інтерфейс користувача

Алгоритм циклічного інтерфейсу користувача характеризується модульністю програми та винесенням основної логіки в підпрограми:

Формальний опис	Приклад

<Блок ініціалізації глобальних змінних>
repeat
  <Блок виведення меню>
 	
  c:=readkey;
  case c of
  <Константа меню>: <Процедура обробки>;
  […]
  <Константа виходу>: break;
  end;
until true;
	s:=0;
	
	repeat
	  clrscr;
	  writeln('Програма для обчислення суми елементів масиву');
	  writeln('1 - введення даних');
	  writeln('2 - порахувати суму');
	  writeln('3 - виведення даних');
	  writeln('0 - вихід');
	  c:=readkey;
	  case c of
	  1: readArray(a);
	  2: s:=calculate(a);
	  3: writeResult(s);
	  0: break;
	  end;
	until true;
	
	clrscr;
	writeln('Дякуємо за використання

<Блок завершення>	програми. Натисніть Enter для виходу.'); readln;

2. Шаблон організації підпрограми

Підпрограма призначена для виконання деякої дії, котра є елементарною в основній програмі. Існує два типи підпрограм: процедура та функція. Процедура виконується як окремий оператор програми та змінює інформацію (якщо це необхідно) шляхом зміни своїх формальних параметрів-змінних. Функція обов'язково повертає якесь значення та використовується тільки у виразах.

Формальний опис	Приклад
Опис процедури	

procedure <Ім'я процедури>[(<Список формальних параметрів>)];
[<Блок визначення локальних констант>]
[<Блок визначення локальних змінних>]
[<Блок визначення локальних підпрограм>]
 	
begin
<Реалізація алгоритму>
end;
	procedure count(const a : Tarr;
	var S : integer);
	
	const
	D = 1;
	var
	i : integer;
	procedure inc(var x : integer)
	begin
	x:=x+D;
	end;
	begin
	S:=0;
	for i:=1 to N do
	if (a[i] > 0) then
	inc(S);
	end;

Використання процедури	
<Ім'я процедури>[(<Список фактичних параметрів>)];	count(a, counter);
Опис функції	

function <Ім'я функції>[(<Список формальних параметрів>)]:<Тип зворотного значення>;
[<Блок визначення локальних констант>]
[<Блок визначення локальних змінних>]
[<Блок визначення
	function getPi: double;
	begin
	getPi:=3,14;
	end;

локальних підпрограм>]
begin
<Реалізація алгоритму>
<Ім'я функції>:=<Значення>;
end;	

Використання функції	
У виразі: <Ім'я функції>[(<Список фактичних параметрів>)]	angleInGrad := angleInRad*180/getPi;

Алгоритм підпрограми повинен мати максимально просту та прозору структуру, звичайно складається з двох блоків:

Формальний опис	Приклад

<Ініціалізація локальних змінних та параметрів>
<Основні обчислення>
	S:=0;
	
	for i:=1 to N do
	if (a[i] > 0) then
	inc(S);

Враховуючи принципи структурного програмування та обмеженість програмного стека, програмісту варто дотримуватися таких правил:

1. Використання глобальних змінних прийнятне тільки в тих випадках, коли змінна несе допоміжне навантаження та не впливає на основний алгоритм програми (наприклад, підрахунок кількості порівнянь та перестановок в процедурі сортування масиву).
2. Використання параметрів-значень допустиме тільки у випадку передавання параметрів, що займають мало пам'яті – порядкові типи даних.
3. У випадку передачі в підпрограму даних великої розмірності варто використовувати параметри-змінні, якщо дані будуть змінюватися, та параметри-константи, якщо дані змінюватися не будуть.
4. Функції здатні повертати тільки примітивні типи даних (порядкові, дійсні, рядкові, множини та вказівники). За необхідності використовувати функції для повернення складних структур даних необхідно використовувати динамічну пам'ять, або позбутися цієї ситуації (наприклад, використовувати процедуру замість функції).

3. Шаблони організації інтерфейсу користувача

Інтерфейс користувача програми – різновид інтерфейсів, в котрому одна сторона представлена людиною (користувачем), інша – програмою. Є сукупністю засобів та методів, за допомогою яких користувач взаємодіє з програмою.

Під сукупністю засобів та методів інтерфейсу користувача програми розуміються:

1. Засоби – повинні бути необхідними й достатніми, бути зручними та практичними, розташованими і скомпонованими розумно та зрозуміло, наслідувати фізіологію людини, не повинні призводити до негативних наслідків для організму та психіки користувача (все це входить до поняття ергономіки). Засоби вміщують:
   1.1. Засоби виведення інформації з програми до користувача – весь доступний діапазон впливу на людину – екрани, дисплеї, проектори, лампочки, динаміки, зумери, сирени, вібромотори та інше.
   1.2. Засоби введення інформації або команд користувачем в програму – множина всіляких пристроїв для контролю стану людини – кнопки, перемикачі, потенціометри, сервоприводи, датчики положення, руху, звуку.
2. Методи: набір правил, закладених розробниками програми, згідно з якими сукупність дій користувача повинна привести до необхідної реакції програми та виконання потрібної задачі – так званий логічний інтерфейс.

Розробляючи стандартне ПЗ, програміст обмежений у виборі засобів інтерфейсу користувача програми стандартним набором. Для програми, що розробляється в середовищі Turbo Pascal, це:

1) засоби введення інформації: клавіатура;
2) засоби виведення інформації: монітор, вбудований динамік.

Використання нестандартних засобів введення / виведення виконується за допомогою звернення до портів введення / виведення та у пропонованому посібнику не розглядаються.

Цей розділ присвячено методам організації інтерфейсу користувача програми. Згідно з принципами структурного програмування, при організації інтерфейсу користувача програмісту варто дотримуватися таких правил:

1. Усе введення інформації повинно бути зрозумілим та зручним для користувача, тобто блоки інформації, що вводиться, повинні бути логічно відокремленими один від іншого, мати невеликий об’єм, супроводжуватися пояснювальною інформацією та обов’язково перевірятися на коректність.

2. Інформація, що вводиться, повинна бути повною, чіткою та інтуїтивно зрозумілою користувачу. Великі блоки інформації повинні бути логічно розділені на підблоки. Після виведення кожного блоку інформації користувачу повинно

бути надано час для сприйняття інформації. Кожний блок повинен уміщуватися на екрані.

Нижче надано декілька шаблонів введення / виведення інформації.

Введення примітивних типів даних з перевіркою на коректність

Цикл repeat...until дозволяє вимагати від користувача введення коректної інформації, при цьому користувачу необхідно зазначати межі коректних даних.

Мовна конструкція

```
const
  Nmax=10;
var
  N : word;
…
write('Введіть розмірність масиву (від 1 до ',
  Nmax, '): ');
repeat
  readln(N);
  if (N>Nmax) or (N<0) then
    write('Ви ввели неправильні дані. Розмірність
    масиву повинна бути від 1 до ', Nmax, '.
    Повторіть введення: ');
until (N<=Nmax) and (N>0);
…
```

Екран

```
…
Введіть розмірність масиву (від 1 до 10): 15
Ви ввели неправильні дані. Розмірність масиву
повинна бути від 1 до 10. Повторіть введення: 7
…
```

Виведення примітивних типів даних

Кожне значення, що виводиться, потребує пояснення.

Мовна конструкція

```
var
  S : longint;
…
writeln('Сума елементів масиву дорівнює ', S);
…
```

Екран

```
…
Сума елементів масиву дорівнює 520
…
```

Введення змінної перерахункового типу даних

Введення змінної перерахункового типу даних виконується шляхом явного перетворення змінної цілого типу даних до змінної перерахункового типу (month: = TMonthes(m-1);).

Масив констант monthNames використовується для виведення варіантів, що доступні користувачу.

Необхідно пам’ятати, що константи перерахункового типу нумеруються з 0, а користувачу зручне введення, починаючи з 1. Отже, при виведенні варіантів

номер виводиться як ord(month)+1, а при перетворенні від введеного значення віднімається 1: TMonthes(m-1).

Мовна конст-рукція

```
type
  TMonthes = (jan, feb, mar, apr, may, jun, jul,
        aug, sep, oct, nov, dec);
var
  month : TMonthes;
  m : byte;
const
  monthNames : array [TMonthes] of String[8] =
         ('Січень', 'Лютий', 'Березень',
         'Квітень', 'Травень', 'Червень',
         'Липень', 'Серпень', 'Вересень',
         'Жовтень', 'Листопад', 'Грудень');
…
writeln('Введіть місяць:');
for month:=Low(TMonthes) to High(TMonthes) do
  write((ord(month)+1):2, ' - ',
  monthNames[month]);
write('Номер місяця: ');
repeat
  readln(m);
  if (m<1) or (m>ord(High(TMonthes))+1) then
    write('Ви ввели неправильні дані. Виберіть
      номер місяця. Повторіть введення: ');
until (m<=ord(High(TMonthes))+1) and (m>0);
month:= TMonthes(m-1);
…
```

Екран

```
…
Введіть місяць:
 1 - Січень
 2 - Лютий
 3 - Березень
 4 - Квітень
 5 - Травень
 6 - Червень
 7 - Липень
 8 - Серпень
 9 - Вересень
10 - Жовтень
11 - Листопад
12 - Грудень
Номер місяця: 20
Ви ввели неправильні дані. Виберіть номер місяця.
Повторіть введення: 9
…
```

Виведення змінної перерахункового типу даних

Якщо необхідне виведення змінної перерахункового типу даних, масив констант з мітками для значень перерахункового типу (monthNames) робить виведення тривіальним.

Мовна конструкція

```
type
  TMonthes = (jan, feb, mar, apr, may, jun, jul,
        aug, sep, oct, nov, dec);
var
  month : TMonthes;
const
  monthNames : array [TMonthes] of String[8] =
         ('Січень', 'Лютий', 'Березень',
         'Квітень', 'Травень', 'Червень',
         'Липень', 'Серпень', 'Вересень',
         'Жовтень', 'Листопад', 'Грудень');
…
writeln('Місяць: ', monthNames[month]);
…
```

Екран

```
…
Месяц: Вересень
…
```

Введення одновимірних масивів

Варіант 1. Введення кожного елемента окремо: користувачу для кожного елемента масиву виводиться запрошення із зазначенням індексу елемента, що вводиться, для того, щоб користувач знав, на якому етапі введення він перебуває.

Мовна конструкція

```
var
  a : array [1..Nmax] of double;
  n, i : word;
…
writeln('Введіть елементи масиву:');
for i:=1 to n do begin
  write('a[', i, ']=');
  readln(a[i]);
end;
…
```

Екран

```
…
Введіть елементи масиву:
a[1]=10
a[2]=3.14
a[3]=.52
a[4]=1.17
a[5]=1000
…
```

Варіант 2. Введення елементів одним рядком: користувач може вводити елементи через пробіл, при цьому необхідно зазначити загальну кількість елемен-

тів, котру буде очікувати програма. Оператор readln після циклу зчитує з буфера клавіатури ознаку кінця введення.

Мовна конст-рукція

```
var
  a : array [1..Nmax] of double;
  n, i : word;
…
writeln('Введіть елементи масиву (1..', n,'):');
for i:=1 to n do
  read(a[i]);
readln;
…
```

Екран

```
…
Введіть елементи масиву (1..5):
10 3.14 .52 1.17 1000
…
```

Виведення одновимірних масивів

Допустимо виводити елементи одновимірного масиву в рядок, при цьому необхідно слідкувати, щоб елементи масиву між собою розмежовувалися.

Мовна конст-рукція

```
var
  a : array [1..Nmax] of double;
  n, i : word;
…
writeln('Елементи масиву:');
for i:=1 to n do
  write(a[i]:8:2);
writeln;
```

Екран

```
…
Елементи масиву:
   10.00    3.14    0.52    1.17 1000.00
…
```

Введення багатовимірних масивів

Введення багатовимірних масивів повинно супроводжуватися виведенням інформації про те, на якій стадії введення знаходиться користувач. Допустимо останній індекс вводити рядком.

Мовна конст-рукція

```
var
  a : array [1..Nmax, 1..Mmax] of integer;
  n, m, i, j : word;
…
writeln(Введіть елементи масиву:');
for i:=1 to n do
  for j:=1 to m do begin
    write('a[', i, ',', j, ']=');
    readln(a[i,j]);
  end;
…
```

Екран	… Введіть елементи масиву: a[1,1]=11 a[1,2]=12 a[1,3]=13 a[2,1]=21 a[2,2]=22 a[2,3]=23 …

Виведення багатовимірних масивів

Виведення двовимірних масивів доцільно виконувати у вигляді квадратної матриці. Масиви, розмірністю більш ніж 2, необхідно логічно розбивати на двовимірні масиви та виводити частково.

Мовна конструкція

```
var
  a : array [1..Nmax, 1..Mmax] of integer;
  n, m, i, j : word;
…
writeln('Елементи масиву:');
for i:=1 to n do begin
  for j:=1 to m do
    write(a[i,j]:4);
  writeln;
end;
…
```

Екран

```
…
Елементи масиву:
  11  12  13
  21  22  23
…
```

ДЛЯ НОТАТОК

Навчальне видання

Ломонос *Ірина Миколаївна*
Ванжа *Лілія Іванівна*
Мацецка *Наталія Артурівна*

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з курсу
ОСНОВИ АЛГОРИТМІЧНИХ МОВ

(Електронне видання)

Редактор А. О. Цяпало
Технічний редактор Т. О. Важеніна

Підписано до друку 27.12.2019
Формат 60 х 84/16. Папір офсетний.
Умовн. друк. арк. 5,58
Зам. 127

Донецький національний університет імені Василя Стуса
21021, м. Вінниця, 600-річчя, 21
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 5945 від 15.01.2018