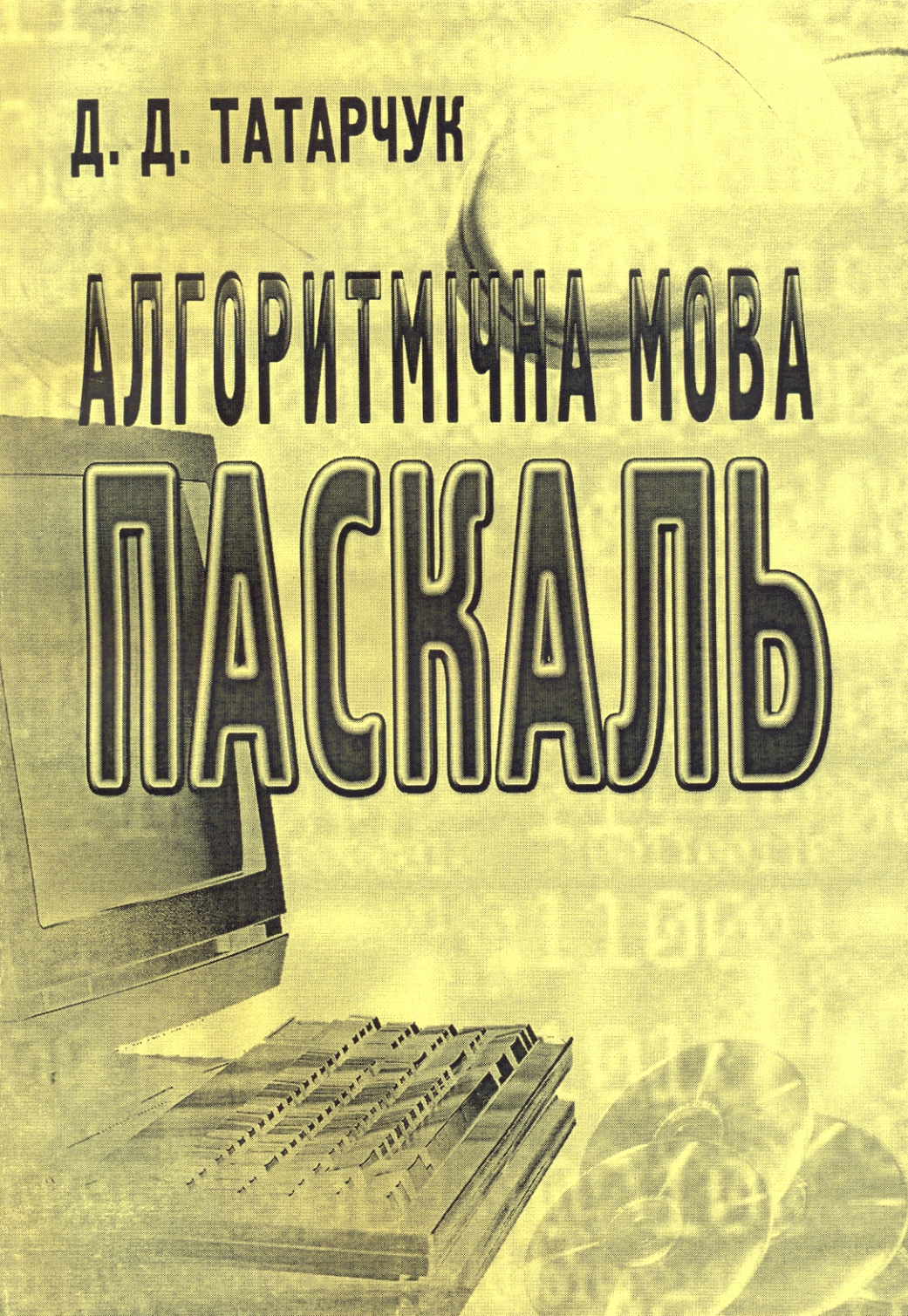
Д. Д. ТАТАРЧУК

# АЛГОРИТМІЧНА МОВА ПАСКАЛЬ



Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Д. Д. Татарчук

# Алгоритмічна мова Паскаль

Київ
НТУУ «КПІ»
2007

УДК 004.43 (075.8)
ББК 32.973.26-018. І я 73
Т23

*Гриф надано Методичною радою НТУУ «КПІ»*
*(Протокол №4 від 21.12.2006 р.)*

Рецензенти:
*Б. П. Бездітний*, канд. фіз.-мат. наук, доц.,
Інститут підготовки кадрів промисловості
Міністерства промислової політики України
*Н. І. Вовкодав*, канд. техн. наук, доц.
Національний університет харчових технологій
*О. В. Кочубей*, канд. техн. наук, доц.
Національний університет харчових технологій

**Татарчук Д. Д.**

Т23 Алгоритмічна мова Паскаль: Навч. посіб. – К.: НТУУ «КПІ», 2007. – 84 с.

**ISBN 978-966-622-254-4**

У посібнику викладено навчальний матеріал з програмування алгоритмічною мовою Паскаль. Розглянуто основні конструкції та типи алгоритмічної мови Паскаль. Особливу увагу приділено структурованим типам даних та методам роботи з ними.

Для студентів бакалаврату напряму підготовки «Електроніка».

**УДК 004.43 (075.8)**
**ББК 32.973.26-018. І я 73**

ISBN 978-966-622-254-4

# ЗМІСТ

# ВСТУП

Мова програмування Паскаль була розроблена швейцарським вченим Нікласом Віртом, як засіб для навчання студентів програмуванню. Однак ця мова виявилась настільки вдалою, що досить швидко завоювала популярність серед програмістів і завдяки корпорації Borland перетворилася в потужну сучасну професійну систему програмування, за допомогою якої можна вирішувати різноманітні задачі від розробки відносно простих обчислювальних програм, до створення складних реляційних систем управління базами даних. Вдале поєднання простоти мови з вражаючими можливостями призвело до появи потужних систем програмування на її основі, таких, як Turbo Pascal, Delphi, Kylix. Особливо привабливою мову Паскаль робить те, що на сьогодняшній день – це одна з небагатьох мов програмування, яка має розвинені засоби програмування для багатьох поширених сучасних операційних систем, що дозволяє досить легко переносити програми, написані на мові Паскаль, до різних операційних систем.

В зв’язку із сказаним вище та з огляду на швидкий розвиток інформаційних технологій оволодіння засобами мови Паскаль стало невід’ємною частиною базової підготовки кваліфікованих спеціалістів, спроможних ефективно використовувати всі можливості сучасної обчислювальної техніки для вирішення практичних задач. Саме тому метою даного посібника є максимальне полегшення процесу вивчення студентами мови Паскаль. При цьому автор не ставив перед собою завдання написати повний довідник по мові Паскаль. Автор ставив перед собою завдання створити простий посібник по програмуванню на мові Паскаль, який би містив матеріал, достатній для використання при розв’язку типових обчислювальних задач, які щоденно доводиться вирішувати студентам технічних вузів.

# 1 ОСНОВНІ ЕЛЕМЕНТИ МОВИ ПАСКАЛЬ

## 1.1 Основні поняття

### 1.1.1 Алфавіт мови

Алфавіт – це сукупність дозволених в мові символів чи груп символів, що розглядаються як єдине ціле. Алфавіт мови Паскаль складається з літер, цифр, спеціальних символів та символів, що не використовуються. Крім того до алфавіту мови Паскаль можна віднести зарезервовані слова та стандартні директиви [1].

До літер відносяться великі та малі літери латинського алфавіту-від A до Z та від a до z. При цьому Паскаль не розрізняє одноіменні великі та малі літери. Також літерою вважається символ підкреслювання “_”.

До цифр відносяться десяткові цифри від 0 до 9 та шістнадцяткові цифри. Кожна шістнадцяткова цифра має значення від 0 до 15. Перші десять цифр позначаються десятковими цифрами, інші шість латинськими буквами від A до F та від a до f. Для того, щоб відрізнити шістнадцяткове число від послідовності символів, перед ним записують символ $.

Спеціальні символи можна умовно розділити на розділові знаки, знаки пунктуації, знаки операцій та зарезервовані слова.

Розділові знаки використовуються для відділення один від одного елементів програми. У якості розділових знаків можна використовувати :

- символ пробілу;
- символ табуляції;
- коментарі.

Кілька символів пробілу, що слідують один за одним вважаються одним.

Коментарем називається будь-яка послідовність символів, розміщена у фігурні скобки:

**{ коментар }**

Замість фігурних скобок можна використовувати пари символів (* та *). Коментарі можуть займати будь-яку кількість строк і ігноруються під час виконання програми. Основне призначення коментарів – пояснення по тексту програми. Не слід плутати коментарі з директивами компілятора, що починаються з пари символів {$ та закінчуються символом }. Директива компілятора розглядається як єдине ціле і призначена для керування процесом компіляції програми.

До знаків пунктуації відносяться:

**( ) (* *) [ ] { } “ , . : ; := .. ^ @ $ #.**

Знаками операцій являються:

**+ - * / = < > <> <= >= .**

До стандартних директив мови Паскаль відносяться такі слова:

| | | | | |
|---|---|---|---|---|
| **absolute** | **assembler** | **external** | **far** | **forward** |
| **interrupt** | **near** | **private** | **virtual .** | |

Зарезервованими у мові Паскаль є такі слова:

| | | | | |
|---|---|---|---|---|
| **and** | **asm** | **array** | **begin** | **Case** |
| **const** | **constructor** | **destructor** | **div** | **Do** |
| **downto** | **else** | **end** | **file** | **For** |
| **function** | **goto** | **If** | **implementation** | **In** |
| **inline** | **interface** | **label** | **mod** | **Nil** |
| **not** | **object** | **Of** | **or** | **packed** |
| **procedure** | **program** | **record** | **repeat** | **Set** |
| **shl** | **shr** | **string** | **then** | **To** |
| **type** | **unit** | **until** | **uses** | **Var** |
| **while** | **with** | **xor** | | |

Стандартні директиви та зарезервовані слова в програмі можуть використовуватися лише за своїм прямим призначенням, і не можуть бути використані у будь-якій іншій якості.

Символи, що не використовуються, це символи які не являються безпосередньо частиною мови Паскаль і можуть бути використані лише у коментарях, а також у символьних та строкових константах. До таких символів належать, наприклад, літери українського алфавіту.

### 1.1.2 Ідентифікатори

Ідентифікатори(імена) використовують для позначення констант, типів, змінних, процедур, функцій, класів модулей, програм і т.і. Ідентифікатори у Паскалі можуть бути будь-якої довжини, але значущими є лише перші 63 символи. Ідентифікатор може складатися з літер латинського алфавіту, цифр та символів підкреслювання, але починатися може лише з букви або символу підкреслювання. Регістр символів значення не має. До складу ідентифікаторів не можна включати спеціальни символи, символи, що не використовуються та символи пробілу. В якості ідентифікаторів не можна використовувати стандартні директиви, зарезервовані слова та імена елементів, що входять до стандартної бібліотеки.

Приклади допустимих ідентифікаторів:

**a, alpha, MyVar, _beta, n123** .

Приклади недопустимих ідентифікаторів:

**5a** – починається з цифри;

**block#** - містить спеціальний символ # ;

**My Var** – містить символ пробілу ;

**mod** – зарезервоване слово.

### 1.1.3 Структура програми на Паскалі

Програма на мові Паскаль складається з речень (**statements**). Кожне речення закінчується символом “**;**” (крапка з комою). Текст програми закінчується символом “**.**” (крапка).

Першим реченням програми є заголовок програми. Заголовок складається з зарезервованого слова **program**, за яким іде ім’я програми. Після імені у скобках можуть бути перелічені параметри програми. Наприклад:

**program(input, output);**

В сучасних діалектах мови програмування Паскаль заголовок розглядається як коментар, а тому є необов’язковим.

Речення програми об’єднуються в розділи. Існує шість видів розділів:

- розділ опису міток;
- розділ опису констант;
- розділ опису типів;
- розділ опису змінних;
- розділ опису процедур та функцій;
- розділ операторів.

Розділ операторів розташовується між зарезервованими словами **begin end**. Після слова **end** , яким закінчується програма ставлять крапку. В цьому розділі розміщують речення, що задають послідовність дій, яку повинен виконати комп’ютер. Всі інші розділи програми носять описовий характер. Будь-який з розділів крім розділу операторів може бути відсутнім, якщо в ньому нема потреби.

Розділ опису міток починаєтья зарезервованим словом **label** , після якого розміщують список міток. Міткою може бути будь-який дозволений ідентифікатор або ціле число без знаку. Мітки дозволяють відмітити любий

оператор, щоб на нього, при необхідності, можна було передати керування з іншого місця програми.

Розділ констант починається зарезервованим словом **const** , після якого розміщують речення типу:

**<ім'я константи>=<значення константи>;**

Розділ опису типів починається зарезервованим словом **type** , після якого розміщують речення типу:

**<ім'я типу>=<опис типу>;**

Розділ опису змінних починається зарезервованим словом **var** , після якого розміщують речення типу:

**<ім'я змінної (або список змінних через кому)>:<тип змінної(змінних)>;**

Розділ опису процедур та функцій не виділяється спеціальними словами, оскільки кожна процедура чи функція має свій власний заголовок, що починається зарезервованим словом **procedure** або **function** відповідно.

В будь-яке місце програми можуть бути включені коментарі. Наявність коментарів не змінює змісту програми і не впливає на її виконання. Всі ідентифікатори, що використовуються у програмі повинні бути описані у відповідному розділі до того, як вони будуть використані у програмі. При необхідності скористатися можливостями стандартної бібліотеки можна підключити потрібний бібліотечний модуль за допомогою зарезервованого слова **uses**.

Для прикладу розглянемо програму обчислення довжини кола l та площі круга s за введеним з клавіатури значенням радіуса r :

```
program demo; {заголовок програми}
uses crt; {підключення стандартного бібліотечного модуля crt }
const     {розділ опису констант}
   pi=3.14; {опишемо константу pi}
var      {розділ опису змінних}
```

```
    r,s,l: real; {опишемо змінні r,s,l дійсного типу }
begin   {розділ операторів}
    clrscr; {очищення екрану за допомогою функції із модуля crt}
    write('Input r = ','? '); {виведемо на екран r = ?}
    read('r'); {отримаємо з клавіатури значення змінної r}
    s:=pi*sqr(r); {обчислимо значення змінних s,l}
    l:=2*pi*r;
    writeln('s=',s);{виведемо отримані результати на екран}
    writeln('l=',l);
end.
```

Результат виконання програми:

**Input r = ? 1**

**s=3.14**

**l=6.28**

## 1.2 Класифікація типів даних у Паскалі

Типи даних в Паскалі можна розділити на стандартні, які є частиною мови Паскаль, та нестандартні, що визначаються програмістом.

Стандартні типи мови Паскаль можна класифікувати наступним чином:

1. Прості типи
    - а). Порядкові типи
        - Цілі
        - Символи
        - Булевскі
        - Перераховні
        - Обмежені
    - б). Дійсні типи
2. Строки

3. Структуровані типи
    а). Множини
    б). Масиви
    в). Записи
    г). Файли
    д). Об’єкти
4. Покажчики
5. Процедурні типи

Стандартні типи не потребують, щоб їх визначали у розділі опису типів. Їх можна одразу використовувати при визначенні змінних у розділі опису змінних.

Нестандартні типи потребують, щоб програміст їх перед використанням визначив. Зробити це можна у розділі опису типів. Докладніше про це буде розказано у розділі 2.

## 1.3 Прості типи

До простих типів у Паскалі відносяться порядкові і дійсні типи. Порядкові типи характеризуються тим, що відповідні їм значення складають скінченну впорядковану множину значень і кожне значення має свій порядковий номер. Значеннями дійсних типів є числа, що мають або можуть мати дробову частину.

### 1.3.1 Порядкові типи

До порядкових типів відносять цілі, логічний(булевський), символьний, перераховний типи та тип діапазон(обмежений тип). Для виразів порядкового типу допустимі такі функції:

- ord(x) – повертає порядковий номер значення данного виразу. Для цілих типів вертає саме значення x, для булевського 1 для true або 0 для false, для символьного - ASCII код символу, для перераховного – порядковий номер елемента (число в діапазоні від 0 до 65535). Для обмеженого типу результат залежить від властивостей базового порядкового типу;

- pred(x) – повертає значення, що передує значенню x;
- succ(x) – повертає значення, яке є наступним після значення x;
- high(x) – повертає максимально можливе значення для типу змінної x;
- low(x) – повертає мінімально можливе значення для типу змінної x.

Відзначимо, що функція pred є невизначеною для першого по порядку значення заданного типу, а функція succ – для останнього.

#### 1.3.1.1 Цілі типи

В Паскалі визначено п’ять стандартних цілих типів: shortint (коротке ціле), integer (ціле), longint (довге ціле), byte (один байт), word (одне машинне слово).

Цілі типи відрізняються діапазоном значень та розміром пам’яті необхідної для зберігання значення даного типу (дивись таблицю 2.1).

*Таблиця* 1.1. Цілі типи

| **Тип** | **Діапазон значень** | **Розмір пам’яті, байт** |
|---|---|---|
| integer | -32768…32767 | 2 |
| shortint | -128…127 | 1 |
| longint | -2147483648…2147483647 | 4 |
| byte | 0…255 | 1 |
| word | 0…65535 | 2 |

#### 1.3.1.2 Символьний тип

Значеннями символьного типу являються коди символиів із множини ASCII (американський стандартний код для обміну інформацією). Ця множина містить 256 впорядкованих символів з кодами від 0 до 255. До їх складу входять цифри, літери, символи псевдографіки та спеціальні керуючі символи. Якщо символ має графічне відображення, то його можна записати в програмі, помістивши це зображення між одинарними кавичками: ‘d’ , ’ ‘ , ‘%’.

Крім цього будь-який символ, у тому числі й керуючий, можна записати у програмі використовуючи його код, що слідує за символом #.

Наприклад:

#37 еквівалентно ‘%’;

#103 еквівалентно ‘g’;

#27 еквівалентно натисканню клавіші <Esc>.

При наявності системної підтримки символьна множина може містити символи російського та українського алфавіту.

#### 1.3.1.3 Обмежений тип

На основі порядкових типів можна створювати обмежені типи. Це робиться шляхом визначення мінімального та максимального значення діапазону.

Наприклад:

```
type
    digit=’0’..’9’;
var
    n:digit;
    letter=’a’..’z’;
```

Обмежений тип успадковує всі властивості базового типу (в тому числі набор допустимих операцій).

#### 1.3.1.4 Перераховний тип

Перераховний тип визначається шляхом перерахування всіх його значень, причому кожне значення визначається символьним ім’ям. Список значень поміщують у круглі скобки.

Наприклад:

```
type
week = ( Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
var
```

day : week;

arrow : (left, up right,down);

Для значень перераховних типів визначено операції порівняння. При цьому вважається, що значення у списку подано у порядку зростання. Імена із списку значень вважаються константами. Недопускається опис двох чи більшого числа типів з константами, що співпадають.

### 1.3.2 Дійсні типи

В Паскалі визначено п’ять дійсних типів: real, single, double, extended, comp, які відрізняються діапазоном, точнічтю та об’ємом пам’яті необхідної для зберігання значень цих типів.

*Таблиця* 1.2. Дійсні типи

| **Тип** | **Діапазон значень** | **Кількість значущих цифр** | **Розмір пам’яті, байт** |
|---|---|---|---|
| real | $\pm(2.9*10^{-39}\ldots1.7*10^{38})$ | 11-12 | 6 |
| single | $\pm(1.5*10^{-45}\ldots3.4*10^{38})$ | 7-8 | 4 |
| extended | $\pm(5.01*10^{-324}\ldots1.7*10^{308})$ | 15-16 | 8 |
| double | $\pm(3.4*10^{-4932}\ldots1.1*10^{4932})$ | 19-20 | 10 |
| comp | $-2^{63}+1\ldots2^{63}-1$ | 19-20 | 8 |

Дійсні значення можуть відображатися в формі з фіксованою крапкою або з плаваючою крапкою.

Наприклад:

347.2 - з фіксованою крапкою;

3.472e2 – з плаваючою крапкою.

В обох випадках записано те саме число.

## 1.4 Покажчики

Покажчик – це змінна, що містить адресу іншої змінної, тобто вказує на неї.

Всі покажчики діляться на типізовані та нетипізовані. Типізовані покажчики вказують на змінні певних наперед визначених типів. Нетипізовані покажчики можуть вказувати на змінні будь-яких типів. Для опису типізованих покажчиків використовується символ ^.

Наприклад:

```
var
    p1:^integer;{покажчик на змінну типу integer}
    p2:^real;{покажчик на змінну типу real}
```

Для опису нетипізованих покажчиків використовують ключове слово pointer.

Наприклад:

```
var
    p:pointer;
```

Нетипізовані покажчики зручно використовувати, якщо треба виконувати якісь операції з адресами змінних різних типів.

Наприклад:

```
var
    p1:^integer;
    p2:^real;
    pp:pointer;
        …
    pp:=p1;
    p2:=pp;{допустима операція а пряме виконання p2:=p1 -недопустиме}
```

Покажчики можуть вказувати на дані любого допустимого у Паскалі типу крім файлового.

Основне призначення покажчиків – це динамічне керування пам’яттю. Як уже говорилося покажчик це змінна, яка містить адресу деякої області пам’яті або змінної. Це дає можливість виділяти додаткову пам’ять при необхідності і звільняти її, коли така необхідність минає. Для виділення пам’яті в Паскалі існують такі процедури :

new(var p:^<тип>) – виділяє пам’ять під змінну заданого типу. Адреса виділеної області пам’яті зберігається у типізованому покажчику p. Покажчик повинен бути попередньо описаний, як покажчик на даний тип;

getmem(var p:pointer;size:word) – виділяє область пам’яті заданого розміру size, для збереження нетипізованих даних. Адреса виділеної області записується у нетипізований покажчик p.

Для звільнення пам’яті, що була виділена за допомогою процедур new та getmem необхідно використовувати наступні процедури:

dispose(var p:^<тип>) – звільнює пам’ять виділену процедурою new. Типи вказані при виділенні і при звільненні пам’яті повинні співпадати;

freemem(var p:pointer; size:word) – звільнює область пам’яті розміром size, виділену процедурою getmem. Розмір size при виділенні пам’яті і при звільненні повинен співпадати.

Крім того для зручності роботи з динамічною пам’яттю в Паскалі є дві функції, що дозволяють перевірити наявність в системі вільної пам’яті, доступної для динамічного виділення:

function maxavail : longint – функція, яка повертає розмір найбільшої неперервної області вільної пам’яті, доступної для динамічного розподілу. Цей розмір відповідає максимальному розміру динамічної змінної, яка може бути розміщена у пам’яті в даний момент часу;

function memavail : longint – функція, яка повертає сумарний розмір вільної пам’яті, доступної для динамічного розподілу.

Використання динамічного розподілу пам’яті дозволяє значно підвищити ефективність використання оперативної пам’яті. Після того, як пам’ять

виділена, над нею можна виконувати ті ж дії, що і над звичайною змінною. Після звільнення пам’яті вона стає доступною для повторного використання при роботі з іншими даними. Для використання області пам’яті, на яку вказує покажчик, застосовують символ ^, що розміщується після імені відповідного покажчика, як це показано у нижченаведеному прикладі:

```
var
  i,j:^integer;{описуємо два покажчики на змінні цілого типу}
  r:^real;{описуємо покажчик на змінну дійсного типу}
  k:integer;
  d:real;
begin
  new(i);{виділяємо область пам’яті під змінну цілого типу і вказує на цю область}
  new(j);{ виділяємо ще одну область пам’яті під змінну цілого типу j вказує на цю область }
  new(r);{ виділяємо область пам’яті під змінну дійсного типу r вказує на цю область }
  i^:=5;{записуємо число 5  у область пам’яті на яку вказує i}
  j^:=i^*2;{ записуємо число 10  у область пам’яті на яку вказує j}
  k:=i^+j^;{k:=5+10;}
  r^:=pi;{записуємо 3.14 до області пам’яті, на яку вказує покажчик r}
  d:=sqrt(r^);{d:=√π ;}
  dispose(r);{звільнення пам’яті на яку вказує r. Ця пам’ять стає доступною для іншого використання}
  dispose(j);{ звільнення пам’яті на яку вказує j. Ця пам’ять стає доступною для іншого використання }
  dispose(i);{ звільнення пам’яті на яку вказує i. Ця пам’ять стає доступною для іншого використання }
end.
```

## 1.5 Константи

В мові Паскаль допустимими є константи трьох типів : звічайні константи, іменовані константи, типізовані константи.

Звичайна константа – це число, символ, строка або логічне значення. Числові константи можуть бути цілими або дійсними, а також додатніми та від'ємними. В дійсних константах ціла частина відокремлюється від дробової за допомогою крапки. Перед від'ємною константою ставиться знак "мінус"(знак "плюс" перед додатньою константою можна не ставити), наприклад:

512 0.1 -522.1

Дійсні константи можуть бути записані у вигляді числа з плаваючою крапкою, тобто у вигляді $\pm ae \pm n$,

де a – число (як правило менше 10 за модулем);

e – спеціальний символ;

n – порядок числа.

Наприклад:

1.28e-3 {значення числа – $1.28 \times 10^{-3}$}

Строкові та символьні константи у програмі виділяють одинарними лапками:

'Input radius' '5.3' 'A'

Строкова константа також може бути записана за допомогою внутрішніх кодів символів, які можна визначити за допомогою таблиці кодів. Перед кодом кожного символа у цьому випадку необхідно ставити спеціальний символ '#'. Так , наприклад строку 'ABC' за допомогою кодів можна записати у вигляді : #65#66#67, де 65 – код символа 'A' , 66 – код символа 'B', а 67 – код символа 'C'.

Логічні константи можуть мати лише значення true (істинне) або false (хибне).

Іменована константа відрізняється від звичайної тим, що вона має ім'я. Це дозволяє використовувати у програмі замість значення константи її ім'я. Це

зручно, коли константа використовується неодноразово або коли константа має велику довжину. Описати іменовану константу можна у розділі опису констант, наприклад:

```
const
    prise = 250;
    pi_num = 3.14;
    name = ‘Ivan’
```

При визначенні іменованої константи в правій частині визначення можуть бути використані звичайні константи, раніше описані іменовані константи, знаки операцій, а також деякі стандартні функції Паскалю :

| | | | | |
|---|---|---|---|---|
| Abs | Chr | Hi | High | Length |
| Lo | Low | Odd | Ord | Pred |
| Round | SizeOf | Succ | Swap | Trunc |

Наприклад :

```
const
    alpha=-1.5;
    beta=abs(alpha);
    name=’Ivan’+’ ‘+’Petrovich’;
```

Використання іменованих констант робить програму більш зрозумілою і спрощує процесс заміни значення констант у програмі.

Типізована константа визначається наступним чином :

<константа>:< тип>=< константний вираз>;

Наприклад:

```
const
     pi:real=3.14;
```

Значення констант обчислюються при компіляції і, не можуть бути змінені під час виконання програми. Вийнятком є типізовані константи, значення яких може бути змінено під час виконання програми, за умови, що була виконана

директива компілятора {$J+}. Якщо ж була виконана директива {$J-}, то типізовані константи перетворюються на іменовані константи, і їх значення не можна змінити під час виконання програми.

Якщо в програмі не встановлено якусь із цих директив явним чином, то діє директива {$J+}.

## 1.6 Змінні

Змінні – це іменовані величини, визначені в програмі, значення яких може змінюватись під час її виконання. В якості значень змінних можуть виступати будь-які стандартні або задані програмістом нестандартні типи, визначені в програмі. Під кожну змінну виділяється область оперативної пам’яті, розмір якої залежить від типу змінної.

Кожна змінна, яка використовується в програмі, повинна бути описана в розділі опису змінних.

Наприклад:

```
var
        x : real;
        i : integer;
        c : char;
        h : boolean;
```

Якщо в програмі є кілька змінних одного типу, то при описі їх можна об’єднати в групу, перерахувавши їх через кому.

Наприклад :

```
var
    x,y,z : real ;
    i, j, k : integer;
    str1, str2 : string;
```

Всі змінні, описані в програмі можна розділити на локальні та глобальні. Змінні, описані в розділі var процедур чи функцій, є локальними. Вони існують лише під час виконання відповідної процедури чи функції. Змінні, описані у розділі var основної програми, є глобальними, вони існують до закінчення виконання програми.

## 1.7 Операції , операнди та вирази

В мові програмування Паскаль для виконання дій над даними існують операції. Операції – це дії, що виконуються над даними (операндами). Операції позначаються спеціальними символами чи групами символів. Наприклад у виразі $x + y$ змінні $x, y$ - це операнди, а "+" – це символ, що означає операцію додавання. Якщо операція застосовується одночасно до двох операндів, то вона називається бінарною (наприклад додавання чи віднімання ), а якщо тільки до одного оператора, то вона називається унарною (наприклад символ "-" у позначенні від'ємного числа). Кожна операція може застосовуватись лише до операндів певних типів. Дозволені в мові програмування Паскаль операції умовно можна розділити на кілька груп :

- арифметичні операції;
- операції відношення;
- логічні (булевські) операції ;
- інші операції.

Арифметичні операції можуть застосовуватись до операндів цілого та дійсного типу. Допустимі арифметичні операції наведено в таблиці 1.3.

*Таблиця* 1.3. Арифметичні операції

| Позначення | Операція | Тип операндів | Тип результату |
|---|---|---|---|
| + | додавання | цілий або дійсний | цілий або дійсний |
| - | віднімання | цілий або дійсний | цілий або дійсний |

| | | | |
|---|---|---|---|
| * | множення | цілий або дійсний | цілий або дійсний |
| / | ділення | цілий або дійсний | дійсний |
| div | цілочисленне ділення | цілий | цілий |
| mod | залишок від ділення | цілий | цілий |

З таблиці 1.3 видно, що результат арифметичних операцій над даними цілого типу є цілим, а результат арифметичних операцій над даними дійсного типу є дійсним. Вийнятком є лише операція ділення, результат якої завжди має дійсний тип. Зміст арифметичних операцій аналогічний відповідним операціям математики і не потребує пояснень за вийнятком операцій цілочисленного ділення та залишку від ділення.

Результатом операції цілочисленного ділення буде ціле число, яке утворюється за рахунок відкидання дробової частини від результату звичайного ділення. Наприклад результатом операції (5 div 3) буде ціле число 1.

Результат операції залишку від ділення (i mod n) може бути обчислений з виразу (i-(i div n)*n). Наприклад в результатом виконання операції (5 mod 3) буде число 2.

Для всіх типів даних дозволені операції відношення. Операції відношення – це бінарні операції, що використовуються для порівняння двох операндів однакового типу, і результат їх виконання має логічний тип :

= - операція перевірки на рівність. Результатом є true , якщо операнди рівні між собою або false в іншому випадку;

<> - операція перевірки на не рівність операндів. Результатом є true , якщо операнди не рівні між собою або false в іншому випадку;

> - операція перевірки чи перший операнд більший за другий. Результатом є true , якщо перший операнд більший за другий, інакше результат - false;

< - операція перевірки чи перший операнд менший за другий. Результатом є true , якщо перший операнд менший за другий, інакше результат - false;

>= - операція більше або дорівнює. Результатом є true , якщо перший операнд більший за другий або дорівнює йому, інакше результат - false;

<= - операція менше або дорівнює. Результатом є true , якщо перший операнд менший за другий або рівний йому, інакше результат - false.

Для даних цілого і логічного типів можуть застосовуватись логічні операції. Допустимі логічні операції наведено у таблиці 1.4.

*Таблиця* 1.4. Логічні операції

| Позначення | Операція | Тип операндів | Тип результату |
|---|---|---|---|
| not | логічне заперечення | цілий,логічний | цілий,логічний |
| and | логічне І, кон'юнкція | цілий,логічний | цілий,логічний |
| or | логічне АБО, диз'юнкція | цілий,логічний | цілий,логічний |
| xor | виключне логічне АБО | цілий,логічний | цілий,логічний |

Перша логічна операція в таблиці 1.4 – це унарна операція. Її дія полягає в тому, що вона змінює логічне значення на протилежне. Наприклад, якщо a-логічна змінна, що має значення true, то результатом операції (not a) буде false.

Всі інші логічні операції – бінарні. Для їх використання необхідно два операнди. Результат дії бінарних логічних операцій при різних значеннях операндів наведено в таблиці 1.5.

Логічні операції можуть застосовуватись також до даних цілого типу (таблиця 1.4). В цьому випадку результатом логічної операції є також ціле число, біти якого формуються із бітів операндів за правилами наведеними в таблиці 1.6.

*Таблиця* 1.5. Результат дії логічних операцій над операндами логічного типу

| Значення операнда 1 | Значення операнда 2 | Результат операції | | |
|---|---|---|---|---|
| | | and | or | xor |
| true | true | true | true | false |
| false | true | false | true | true |
| true | false | false | true | true |
| false | false | false | false | false |

*Таблиця* 1.6. Правила побітової дії логічних операцій

| Значення біту операнда 1 | Значення біту операнда 2 | Результат операції | | |
|---|---|---|---|---|
| | | and | or | xor |
| 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

Результатом дії операції not на ціле число буде ціле число, біти якого зміняться на протилежні (нулі стануть одиницями, а одиниці нулями). Розглянемо приклади:

a:= not 1; {a=254}

a:=1 or 2; {a=3}

a:=1 and 3; {a=1}

Окрім арифметичних та логічних операцій в Паскалі дозволені операції зсуву, наведені в таблиці 1.7.

*Таблиця* 1.7. Операції зсуву

| Позначення | Операція | Тип операндів | Тип результату |
|---|---|---|---|
| shl | зсув вліво | цілий | цілий |
| shr | зсув вправо | цілий | цілий |

Операції зсуву – бінарні операції. Перший операнд визначає величину, у якій потрібно виконати побітний зсув, а другий операнд вказує величину цього зсуву . При цьому біти, що звільнюються заповнюються нулями, а значення бітів, що вийшли за межі початкової величини втрачується.

Наприклад:

```
b:=5 shl 2; {b=20}
a:=1;
b:=255 shl a; {254}
b:=255 shr 1; {127}
b:=255 shr 9; {0}
```

Інші операції буде розглянуто в главах, присвячених відповідним типам даних.

Операнди і операції формують вирази. У найпростішому випадку вираз може складатися лише з однієї унарної операції. Операції у виразі виконуються у порядку їхнього приорітету (див. таблицю 1.8).

*Таблиця* 1.8. Пріоритет операцій

| Оператори | Пріоритет | Категорія |
|---|---|---|
| @, not | перший (вищий) | унарні операції |
| *,/,div,mod, nd,shl,shr | другий | операції множення |
| +,-,or,xor | третій | операції додавання |
| =, <>, <, >,<=, >=, in | четвертий (вищий) | операції відношення |

Операції з однаковим пріоритетом виконуються зліва направо. Для зміни порядку виконання операцій використовують круглі дужки, оскільки операції, що містяться в дужках, виконуються в першу чергу (мають вищий пріоритет).

## 1.8 Оператори

Основне призначення програми – це виконання деяких дій по обробці данних. Для опису цих дій призначені оператори. Їх можна умовно розділити на прості та складні [2].

Прості оператори – це оператори, які не містять інших операторів. До простих операторів відносяться: оператор привласнення, оператор процедури, оператор безумовного переходу, порожній оператор. До складних операторів відносяться: складений оператор, умовний оператор, оператор вибору, оператори циклів та оператор приєднання.

Оператори можна відмічати мітками і посилатись на них за допомогою операторів безумовного переходу. Кожен оператор закінчується символом крапка з комою “;”.

Умовний оператор призначений для реалізації розгалудженного обчислювального процесу. В загальному випадку оператор записується у вигляді:

if <вираз> then <оператор1> else <оператор2>;

Спочатку обчислюється вираз. Вираз повинен повертати значення логічного типу. Якщо це значення дорівнює true, то виконується оператор1, якщо false, то виконується оператор2.

Наприклад:

if(x<5) then x:=x+1 else x=5;

Крім того умовний оператор може використовуватись у скороченій формі:

if <вираз> then <оператор> ;

Наприклад:

if(x<>0) then y:=1/x;

При використанні скороченної форми оператор буде виконано лише в тому разі, коли результатом обчислення виразу буде true.

Порожній оператор не виконує ніяких дій, до нього не входять ніякі символи. Наприклад, якщо в програмі є два поспіль символи “крапка з комою”(“;”) або крапка з комою знаходиться перед зарезервованим словом end, то це означає, що між ними знаходиться порожній оператор.

Порожній оператор використовується тоді, коли за синтаксисом якогось із складних операторів в даному місці програми необхідна наявність простого оператора, а за логікою програми не потрібне виконання яких-небудь дій.

Оператор привласнення призначений для заміни поточного значення змінної на нове.

В загальному випадку він має вигляд:

<змінна>:=<вираз>;

Наприклад:

a:=b+c;

x:=(a<b) and (b<>0);

s:=’Pascal’;

Вираз повинен формувати значення того ж типу, що й змінна.

Оператор безумовного переходу передає керування оператору, поміченому вказаною міткою. Синтаксична схема оператора має вигляд:

goto <мітка>;

В мові Паскаль мітка – це ціле число від 1 до 9999. В мові Turbo Pascal дозволяється використовувати в якості міток ідентифікатори. При використанні операторів безумовного переходу необхідно дотримуватись таких правил:

- всі мітки, які використовуються в програмі, повинні бути описані у розділі опису міток, причому кожна лише один раз, за синтаксичною схемою label <мітка>;
- мітка, що вказана у операторі безумовного переходу, повинна знаходитись у тому ж блоці чи модулі, що і сам оператор. Іншими словами не дозволяються переходи із процедур чи функцій або переходи всередину процедур та функцій;
- не рекомендуються переходи в середину структурного оператора, оскільки це може викликати непередбачувані наслідки, хоча компілятор не видає попередження в такій ситуації.

Наведемо приклад використання оператора безумовного переходу:

```
  …
label lb;
  …
begin
  …
goto lb;
  …
lb: writeln('Hello!');
  …
end.
```

По можливості слід уникати використання оператора безумовного переходу, оскільки він ускладнює розуміння програми , робить її заплутаною.

Складений оператор – це сукупність операторів, розташованих між ключовими словами begin … end, разом з цими ключовими словами. Складений оператор інтерпретується компілятором як один оператор і використовується тоді, коли за синтаксисом Паскалю в даному місці програми може знаходитись лише один оператор, а за логікою програми необхідно виконати кілька операторів.

Наприклад:

```
if(i<>0) then
        y:=y+1/i
else
        begin
            i:=2;
            y:=y+1/i;
        end;
```

Оператор варіанта використовується в тих випадках, коли необхідно вибрати один варіант серед кількох альтернативних. Синтаксис оператора варіанта має вигляд:

```
case <вираз> of
        <список констант 1>: <оператор1>;
        < список констант 2>: < оператор 2>;
            …
        < список констант N>: < оператор N>;
        else
            <оператор>;
end;
```

Вираз повинен мати порядковий тип розміром один або два байти. Так, наприклад, строковий тип або тип LongInt є недопустимими типами для виразу в операторі варіанта.

При використанні оператора спочатку обчислюється вираз. Потім послідовно перевіряються списки констант. Якщо у якомусь із списків міститься константа, що дорівнює результату обчислення виразу, то виконуєтья відповідний оператор. Якщо жоден список не містить відповідної константи, то виконується оператор після ключового слова else. Ключове слово else може бути опущеним, тоді, в разі відсутності відповідної константи, не буде виконано жодного з операторів.

Наприклад:

```
case i of
    '+': z:=x+y;
    '-': z:=x-y;
end;
…
case dig of
    0,2,4,6,8: writeln('парна цифра');
    1,3,5,7,9: writeln('не парна цифра');
    else
            writeln('не цифра');
end;
```

При написанні програм досить часто виникає необхідність циклічного використання деяких дій (задану кількість разів або до виконання певної умови). Для цього у Паскалі існують оператори циклів. Їх є три види: оператор циклу з параметром, оператор циклу з передумовою та оператор циклу з післяумовою.

Оператор циклу з параметром використовується тоді, коли наперед відомо скільки разів необхідно виконати вказані дії. Він може використовуватись у двох формах:

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

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

В першому випадку спочатку обчислюються початкове і кінцеве значення змінної, і змінна отримує початкове значення. Потім значення змінної порівнюється з кінцевим значенням, якщо її значення менше за кінцеве значення або рівне йому то виконується оператор. Потім значення змінної збільшується на одиницю, і знову порівнюється з кінцевим значенням. Якщо це значення не перевищує кінцеве значення то знову виконується оператор. Коли значення змінної перевищує кінцеве значення, цикл завершується. У другому

випадку значення змінної буде зменшуватись на одиницю і цикл завершиться, коли значення змінної стане меншим за кінцеве значення.

Наприклад:

```
…
f:=1;
for i:=5 downto 2 do f:=f*i;
…
f:=1;
for i:=2 to 5 do f:=f*i;
```

В даному прикладі двома способами обчислюється факторіал числа п’ять.

Оператор циклу з передумовою виконується тоді, коли для виконання деяких дій необхідно попередньо виконати задану умову. Він має синтаксис:

while <вираз> do <оператор>;

Вираз повинен повертати значення булевського типу. Обчислення виразу відбувається до того, як буде виконано оператор. Оператор виконується повторно доти, доки вираз приймає значення true. Для прикладу обчислимо факторіал числа п’ять за допомогою циклу з передумовою:

```
i:=5;
f:=1
while (i>=2) do
begin
    f:=f*i;
    i:=i-1;
end;
```

Оператор циклу з післяумовою має формат:

repeat <оператор1>; <оператор2>; … <операторn> until <умова>;

Оператор циклу з після умовою виконується таким чином: спочатку виконуються оператори, а потім перевіряється умова. Оператори будуть циклічно виконуватись доти, доки умова не стане істинною.

Наприклад:

```
f:=1;
i:=5;
repeat
    f:=f*i;
    i:=i-1;
until (i<2);
```

В наведеному прикладі за допомогою циклу з післяумовою обчислено факторіал числа п’ять.

## 1.9 Процедури та функції. Процедурні типи

Досить часто процес розв’язку складної задачі можна розбити на послідовність розв’язків більш простих задач. Якщо вважати, що програма призначенна для виконання якоїсь складної задачі, то процедури та функції призначені для розв’язку окремих підзадач.

Структура процедур та функцій подібна структурі програми. Процедури та функції мають спільну назву – підпрограми. Кожна підпрограма визначається лише один раз, а використовуватись може багаторазово. Використання підпрограм дозволяє зменшити число повторів однакових послідовностей операторів, зменшити розмір програми, а також прискорити процес програмування і спростити структуру програми.

В програмі опис підпрограм, як правило, здійснюють у розділі опису процедур та функцій. Цей розділ повинен знаходитись до першого використання підпрограм, описаних у ньому. В програмі може бути кілька таких розділів. Зазвичай цей розділ розміщують між розділом опису змінних та розділом операторів.

Функція відрізняється від процедури тим, що функція в результаті своєї роботи повертає значення, яке може бути привласнене змінною, а процедура тільки виконує задану послідовність дій і не повертає ніякого значення. Так наприклад стандартна процедура clrscr очищує екран монітора, а стандартна функція sin обчислює значення синусу заданого кута і повертає обчислене значення у вигляді числа, яке може привласнюватись змінною відповідного типу ( x:= sin(3.14));

Опис процедури має вигляд:

procedure <ім’я процедури>(<список формальних параметрів>);

Опис функції має вигляд:

function <ім’я функції>(<список формальних параметрів>): <тип результату>;

Наприклад:

```
procedure out_put(x,y:integer);
function factorial(a:integer):integer;
```

Визов процедури здійснюється за допомогою речення, що має вигляд:

<ім’я процедури>(<список фактичних параметрів>);

Визов функції може здійснюватись одним із двох способів:

<ім’я функції>(<список фактичних параметрів>);

<ім’я змінної>:=<ім’я функції>(<список фактичних параметрів>);

Перший спосіб використовується тоді, коли обчислене значення не потрібне для подальшого використання, а другий тоді, коли обчислене значення планується використовувати.

Наприклад:

```
var
    a,b,c:integer;
    a:=3;
    b:=5;
    …
out_put(5,3);
```

```
out_put(a,b)
writeln(factorial(5));
a:=factorial(5);
c:=factorial(b);
```

Між формальними та фактичними параметрами повинна бути точна відповідність, тобто кількість, типи і порядок слідування фактичних параметрів повинні співпадати з кількістю типами і порядком слідування формальних параметрів у описі підпрограми.

Як уже відзначалося, при визові підпрограм їм передаються параметри. Параметри можуть передаватися по значенню чи по посиланню. Основна відмінність між ними в тому, що при передачі параметрів по посиланню при закінченні підпрограми у відповідній змінній зберігається значення обчислене при виконанні процедури, а при передачі по значенню, всі зміни, що відбулися із даною змінною у підпрограмі, пропадають після її закінчення і змінна набуває значення, яке вона мала до виконання підпрограми.

Для позначення змінної, що передається по посиланню, в описі функції перед цією змінною ставиться ключове слово var.

Наприклад:

```
procedure primer(x,y:integer; var b:real);
```

Якщо підпрограма змінює значення формального параметра, переданого за посиланням або глобальної змінної, то кажуть, що підпрограма має побічний ефект.

Серед операторів, що входять до складу функції обов’язково повинен бути хочаб один оператор привласнення, в лівій частині якого стоїть ім’я даної функції. Цей оператор визначає значення, яке повертається функцією.

Всі типи , що використовуються при передачі параметрів у підпрограму повинні бути попередньо описані.

Наприклад:

```
program demo;
uses crt;
var
n, fact:integer;

procedure print_results( n,fact:integer);
begin
      writeln(‘factorial(n):=’, fact);
end;

function factorial(n:integer):integer;
var
    i,f:integer;
begin
f:=1;
for i:=2 to n do
      f:=f*i;
factorial:=f;
end;

begin
    clrscr;
    write(‘input n=’);
    readln(n);
    fact:=factorial(n);
    print_results(n,fact);
    readkey;
    end.
```

Для зручності роботи Турбо-Паскаль містить велику кількість власних (“стандартних”) процедур та функцій. Наприклад для виконання арифметичних обчислень з дійсними і цілими числами в Паскалі є ряд арифметичних функцій:

abs - обчислює модуль аргументу;

arctan - обчислює арктангенс аргументу;

cos - обчислює косинус аргументу;

txp - обчислює експоненту аргументу;

frac - обчислює дробову частину аргументу;

int - обчислює цілу частину аргументу;

ln - обчислює натуральний логарифм аргументу;

pi - повертає значення числа $\pi$ {3.1415926535897932385};

sin - обчислює синус аргументу;

sqr - обчислює квадрат аргументу;

sqrt - обчислює квадратний корінь аргументу.

Основні стандартні процедури та функції містяться у файлах стандартних бібліотечних модулів, про які буде сказано пізніше. Повний перелік функцій можна знайти у файлах допомоги системи Турбо-Паскаль.

Мова Паскаль не містить функцій для обчислення показникової функції і функції піднесення до степені. Проте для їх обчислення можуть бути використані експоненціальна і логарифмічна функції - $x^y = e^{y\ln(x)}$.

Турбо-Паскаль розглядає процедури і функції як об’єкти, які можна використовувати в якості значень змінних, параметрів інших функцій і т.д. Це можливо завдяки існуванню у Турбо-Паскалі процедурних типів.

Синтаксис визначення процедурного типу дещо схожий на синтаксис визначення процедури чи функції. Наприклад:

```
type
    proc=procedure;{тип процедура без аргументів}
```

```
    arg_proc=procedure(x,y:integer);{тип процедура з двома
    аргументами}
    math_fun=function(x:real):real;{тип функція}
var
    a,b:proc;{a,в можуть приймати значення процедури без аргументів}
    c:arg_proc;{c може приймати значення процедури з двома
    аргументаим}
    d:math_fun;{d визначається, як функція з одним параметром типу
    real, що повертає значення типу real }
```

Після визначення процедурної змінної вона може бути зв’язана з конкретною процедурою чи функцією, яка має ідентичне визначення, тобто такі ж параметри та тип значення, яке повертається (для функції). Фактично, після присвоєння значення процедурна змінна стає начебто псевдонімом функції і її ім’я може бути використано для визову заданої процедури. Для роботи з процедурними змінними у програму необхідно включити директиву {$F+}. Розглянемо приклад:

```
uses crt;

type
   func=function(x:real):real; {визначаємо процедурний тип func, як функцію,
   що приймає один аргумент типу real  і повертає значення типу real }
var
   fun:func; {описуємо змінну типу func }
 {$F+} {дозволяємо використання процедурних змінних}

function sin1(x:real):real;
begin
   sin1:=sin(x)+1;
end;
```

```
function cos1(x:real):real;
begin
   cos1:=cos(x)+1;
end;

begin
   ClrScr;
  fun:=sin1; {пов’язуємо функцію sin1 з процедурною змінною fun }
   writeln(fun(pi/2)); {обчислюємо значення функції sin1 і виводимо
   обчислене значення на екран, використовуючи процедурну змінну fun }
  fun:=cos1; { пов’язуємо функцію cos1 з процедурною змінною fun }
  writeln(fun(pi/2)); { обчислюємо значення функції cos1 і виводимо
обчислене значення на екран, використовуючи процедурну змінну fun }
   readkey;
end.
```

Як видно із наведеного прикладу наявність механізму процедурних змінних дозволяє маніпулювати процедурами і функціями як звичайними змінними. На перший погляд це нічого не дає програмістові і лише ускладнює програму. Однак це не так. Завдяки існуванню цього механізму можна робити програми значно ефективнішими. Наприклад, якщо нам треба побудувати просте користувацьке меню програми, то без використання процедурних типів нам буде необхідно написати досить складний код з великою кількістю операторів if, або досить складним оператором case, які б дозволили проаналізувати вибір користувача і виконати дії відповідно до того, яку клавішу натиснув користувач. Такий код дуже громіздкий і при великій кількості можливих варіантів дросить повільно працює. Використання ж процедурних типів дозволяє зробити цей код більш простим та ефективнішим:

```
uses crt;
type
   proc=procedure;{визначаємо процедурний тип proc як процедуру без
   аргументів}
var
  cons_proc:array[ord('a')..ord('d')] of proc;{визначаємо масив із чотирьох
  елементів з номерами, що відповідають кодам символів ‘a’,’b’,’c’,’d’(див.
  п. 2.1) }
  c,i:integer;
 {$F+}{дозволяємо використання процедурних змінних}
{визначаємо процедури, що будуть виконуватись при натисканні клавіш, що
відповідають символам  ‘a’,’b’,’c’,’d’ (відповідно a – a_set, b – b_set …)}
procedure a_set;
begin
   writeln('a_set');
   writeln('press any key');
   readkey;
end;
procedure b_set;
begin
   writeln('b_set');
    writeln('press any key');
   readkey;
end;

procedure c_set;
begin
   writeln('c_set');
    writeln('press any key');
```

```
   readkey;
end;

procedure d_set;
begin
   writeln('d_set');
   halt;
end;
{основна програма}
begin
clrscr;
{зв’язуємо відповідні процедурні змінні з потрібними процедурами}
cons_proc[ord(‘a’)]:=a_set;
cons_proc[ord(‘b’)]:=b_set;
cons_proc[ord(‘c’)]:=c_set;
cons_proc[ord(‘d’)]:=d_set;
{виводимо меню на екран і виконуємо процедури меню у відповідності до натиснутої клавіші}
   repeat
      clrscr;
      writeln('a-a_set');
      writeln('b-b_set');
      writeln('c-c_set');
      writeln('d-quit');
      writeln;
      c:=ord(readkey);
      if((c>=97)and(c<=100)) then   cons_proc[c];
   until false;
end.
```

## Контрольні запитання

1. На які групи можна розділити набір символів мови Паскаль?.
2. Чи можна використовувати зарезервовані слова у якості ідентифікаторів?
3. Що являє собою ідентифікатор?
4. Які з наведених ідентифікаторів допустимі при програмуванні на алгоритмічній мові Паскаль: var, my var, my_var,var#? Чому?
5. Чи обов'язкова наявність заголовку програми у мові Паскаль?
6. Що таке константа?
7. Що таке змінна?
8. Чим відрізняється змінна від константи?
9. Чим відрізняється константа від змінної?
10. Що таке типізована константа?
11. Чим відрізняється типізована константа від звичайної константи?
12. З яких розділів складається програма на мові Паскаль?
13. Що таке мітка? Для чого потрібні мітки?
14. Як описати мітку у програмі на Паскалі?
15. Для чого використовується і де розміщується ключове слово uses?
16. Які форми запису чисел використовуються у Паскалі?
17. Як можна класифікувати стандартні типи мови Паскаль?
18. Які типи відносяться до порядкових типів?
19. Які є процедури і функції у мові Паскаль для роботи з порядковими типами?
20. Які є в Паскалі стандартні цілі типи? Наведіть їх характеристики (діапазон значень та розмір).
21. Як у програмі на мові Паскаль описуються цілі змінні?
22. Які операції у Паскалі є допустимими при роботі з цілими типами?
23. Що являє собою символьний тип у Паскалі?
24. Як у програмі на мові Паскаль описуються символьні змінні?
25. Що являє собою обмежений тип у Паскалі?
26. Як описати змінну обмеженого типу?
27. Що являє собою перераховний тип у Паскалі?
28. Як описати змінну перераховного типу?
29. Які є в Паскалі стандартні дійсні типи? Наведіть їх характеристики (діапазон значень та розмір).
30. Які операції допустимі при роботі з дійсними типами?
31. Як описуються у програмі на Паскалі змінні дійсних типів?
32. Що таке покажчик? Чим відрізняються покажчики від інших типів?
33. Як описати покажчик у програмі?
34. Які операції у Паскалі є допустимими при роботі з покажчиками?
35. Які є в Паскалі процедури та функції для роботи з пам'яттю?
36. Що таке іменована константа? Для чого потрібні іменовані константи?
37. Чи може бути змінено значення типізованої константи під час виконання програми? Якщо да, то у яких випадках?

38. Які змінні називаються локальними, а які глобальними?
39. Чим відрізняєтья визначення локальної змінної від визначення глобальної змінної?
40. Що таке операнди?
41. Що таке операції?
42. Які є операції у Паскалі?
43. Що таке вираз?
44. До яких типів операндів можуть бути застосовані арифметичні операції?
45. Який зв'язок між типом операндів і типом результату операції?
46. Які є у Паскалі операції відношення?
47. Що є результатом операцій відношення?
48. Що таке логічна операція? Які є логічні операції у Паскалі?
49. Які особливості застосування логічних операцій до даних цілого типу?
50. Які є у Паскалі операції зсуву? Для чого вони використовуються?
51. Який приорітет операцій у Паскалі? Чи можна змінити приорітет операцій?
52. Що таке оператор? Які бувають оператори у Паскалі?
53. Що таке умовний оператор? Яке його призначення?
54. Який синтаксис умовного оператора?
55. Чому буде дорівнювати змінна a після виконання наведеної нижче послідовності операторів?:

```
b:=3;

if(b>=2) then a:=1 else a:=5;
```

56. Що таке порожній оператор? Коли він використовується?
57. Що представляє собою оператор привласнення? Для чого він використовується?
58. Для чого використовується оператор безумовного переходу?
59. Який синтаксис оператору безумовного переходу у Паскалі?
60. Який оператор буде виконано одразу після оператору безумовного переходу у наведеній нижче послідовності операторів, яке значення матиме змінна a у кінці послідовності?

```
a:=3;

goto    next_step;

a:=a+1;

next_step:    a:=a/2;
```

61. Які існують обмеження у Паскалі при використанні оператора безумовного переходу?
62. Що таке складений оператор? Для чого він використовується?
63. Який синтаксис складеного оператора?

64. Що представляє собою оператор варіанта? Коли він використовується?
65. Який синтаксис оператора варіанту?
66. Яке значення матиме змінна a після виконання наведеної нижче послідовності операторів?

```
b:=7;
case b of
        3: a:=1;
        5: a:=2;
else
        a:=5;
end;
```

67. Які є в Паскалі оператори циклів? Для чого вони використовуються?
68. Який синтаксис оператора циклу з параметром?
69. Який синтаксис оператора циклу з передумовою?
70. Який синтаксис оператора циклу з післяумовою?
71. Яке значення матиме змінна a після виконання кожної з трьох наведених нижче послідовностей операторів?

а)

```
a:=1;
for i:=5 downto 2 do a:=( i-a)*i;
```

б).

```
i:=5;
a:=1;
while (i>=2) do
begin
        a:=(a-i)*a;
        i:=i-1;
end;
```

в).

```
a:=1;
i:=5;
repeat
```

```
    a:=(a+i)*i;
    i:=i-1;
  until (i<2);
```

72. Що таке процедура?
73. Що таке функція? Чим вона відрізняється від процедури?
74. Що таке параметри?Які бувають параметри?
75. Яка різниця між формальними та фактичними параметрами?
76. Які є способи передачі параметрів у Паскалі?
77. Яка різниця між передачею параметрів по значенню та по посиланню?
78. Яка різниця в описі параметрів при передачі по значенню і по посиланню?
79. Що таке стандартні процедури та функції?
80. Які стандартні процедури та функції Вам відомі?
81. Що таке процедурний тип? Для чого він використовується?
82. Як описати у програмі змінну процедурного типу?
83. Яка директива повинна бути включена до програми на Паскалі, щоб дозволити використання процедурних типів?

# 2 СТРУКТУРОВАНІ ТИПИ ДАНИХ У ПАСКАЛІ

## 2.1 Тип масив

### 2.1.1 Поняття масиву

Масив – це набір однотипних елементів. Кожен елемент має свій номер (індекс). Всі елементи масиву упорядковані за своїм індексом. Масив може бути одновимірним чи багатовимірним. Вимірність масивів – не обмежується, але сумарна величина масиву не може перебільшувати 65520 байт.

Задати тип массив можна двома способами:

- з використанням ключового слова type;
- без використання ключового слова type.

Перший спосіб використовується тоді, коли є необхідність багаторазового використання данного типу для опису змінних у ході виконання програми (коли змінні цих типів необхідно передавати до процедур чи функцій, або коли необхідно описати дані такого типу, як локальні в кількох різних процедурах чи функціях).

Наприклад:

а). з використанням ключового слова type

```
type
    massiv = array [1..5] of real;{задано одновимірний масив із 5-ти
                                   елементів}
    matrix = array [1..5,1..5] of real;{задано двовимірний масив(матрицю),
                                   що складається із 5-ти строк, кожна з
                                   яких містить 5 елементів}
…
var
    a,b:massiv;
```

```
    c:matrix;
```

б). без використання ключового слова type

```
var
    a,b: array [1..5] of real;
    c: array [1..5,1..5] of real;
```

Оскільки елементи масиву упорядковані за своїми індексами, то для доступу до конкретного елементу масиву використовується індекс(номер елемента).

Розглянемо приклад:

```
var
  a,b:array[1..5] of real;{створимо 2-а масиви по 5 елементів}
  c:array[1..5,1..5] of char;{створимо матрицю символів розмірністю 5×5}
  …
  a[1]:=3;{присвоїмо значення три першому елементу масиву}
  a[5]:=1;{присвоїмо число 1 п’ятому елементу масива}
  b[3]:=a[5];{третьому елементу масиву b присвоїмо значення п’ятого
        елемента масиву a}
  c[2,3]:=’d’;{третьому елементу другої строки матриці c присвоїти
        значення символа ‘d’}
```

При роботі з масивами зручно користуватись операторами циклів. Наприклад, якщо в масиві цілих чисел mas із десяти елементів значення кожного з елементів повинно дорівнювати квадрату індексу цього елементу, то присвоїти ці значення елементам масиву можна за допомогою наступних команд:

```
var
  mas:array[1..10] of integer;
  i:integer;
  …
```

```
for i:=1 to 10 do
        begin
              mas[i]:=i*i;
        end;
```

### 2.1.2 Обробка масивів

При роботі з масивами найчастіше доводиться виконувати операції сортування та пошуку даних. Від ефективності реалізації цих операцій часто залежить ефективність всієї програми, тому розглянемо ці операції докладніше.

Сортування – це процес перегрупування даних у деякому заданому порядку. Основна мета сортування – полегшити пошук потрібної інформації у заданій послідовності даних[3].

Методи сортування масивів можна розбити на три категорії:

1. Сортування за допомогою включення.
2. Сортування за допомогою вибору.
3. Сортування за допомогою обміну.

Розглянемо вказані методи. Для визначенності будемо вважати, що необхідно виконати сортування у порядку зростання елементів масиву, і всі приклади, що будуть розглянуті відповідатимуть саме такому порядку сортування. Читачеві ж рекомендуємо для тренування модифікувати наведені програми для сортування у зворотньому напрямку.

При сортуванні за допомогою включення елементи умовно ділять на вже відсортовану послідовність $a_1 \ldots a_{i-1}$ і вихідну послідовність $a_i \ldots a_n$. На кожному кроці починаючи з $i=2$ та збільшуючи кожен раз $i$ на одиницю, із вихідної (початкової) послідовності видобувається $i$-тий елемент і перекладається в уже готову послідовність. При цьому він вставляється в потрібне місце готової послідовності. В процесі пошуку потрібного місця чергуються операція порівняння данного елемента з елементами послідовності та операція переміщення по послідовності, тобто вибраний елемент порівнюється з черговим елементом $a_j$ , а тоді $a_i$ або вставляється на місце елемента $a_j$ (якщо

виконано критерій сортування), тоді відповідно $a_j$ зміщується на одну позицію вправо або $a_i$ порівнюється з наступним елементом $a_{j-1}$ (якщо критерій сортування не виконано), при цьому $a_j$ знову ж таки зміщується на одну позицію вправо. Нижче наведено приклад сортування методом включення масиву восьми цілих чисел . Для демонстрації на екран виводяться вихідний масив, масив після кожного етапу вставки чергового елемента на потрібне місце та результуючий масив.

Приклад:

```
 uses crt;
const
    n=8;
var
 i,j,k:integer;
 x:integer;
   a:array[1..n] of integer;
 begin
    clrscr;
    randomize;
    for i:= 1 to n do
    begin
       a[i]:=random(256);
       write (' ',a[i]);
    end;
    writeln;
    writeln;
    for i:=2 to n do
    begin
        x:=a[i];
        j:=i;
```

```
        while x < a[j-1] do
            begin
                a[j]:=a[j-1];
                j:=j-1;
            end;
        a[j]:=x;
        for k:=1 to n do
        write(' ',a[k]);
        writeln;
    end;
    writeln;
    for i:=1 to n do
       write(' ',a[i]);
    readkey;
  end.
```

Сортування за допомогою прямого вибору базується на нижчеперелічених операціях:

1. Обирається елемент з найменшим значенням.
2. Цей елемент обмінюється місцями з першим елементом.
3. Потім п.1-2 повторюються з елементами від 2-го до n-го, потім від 3-го до n-го і т. д.

Наведемо приклад сортування методом прямого вибору масиву восьми цілих чисел . Як і в попередньому прикладі для демонстрації на екран виводяться вихідний масив, масив після кожного етапу вибору і обміну та результуючий масив.

Приклад:

```
uses crt;
const
```

```
  n=8;
var
  i,j,k,z:integer;
  x:integer;
  a:array[1..n]of integer;
begin
   clrscr;
   randomize;
   for i:= 1 to n do
   begin
      a[i]:=random(256);
       write (' ',a[i]);
    end;
    writeln;
    writeln;
    for i:=1 to n-1 do
    begin
       k:=i;
       x:=a[i];
       for j:= i+1 to n do
       begin
          if a[j] < x then
          begin
             k:=j;
             x:=a[k];
          end;
       end;
       a[k]:=a[i];
       a[i]:=x;
```

```
        for z:=1 to n do
            write(' ',a[z]);
        writeln;
    end;
    writeln;
    for i:=1 to n do
        write(' ',a[i]);
    readkey;
end.
```

Сортування за допомогою обміну базується на процесі порівняння і при необхідності обміну місцями двох сусідніх елементів масиву. Ці операції повторюються доти, доки не буде упорядковано весь масив. Треба відмітити, що після першого проходу по всьому масиву максимальний елемент переміщається в крайнє праве положення, і на наступному етапі немає сенсу перевіряти весь масив. Тому на практиці при першому проході перевіряють елементи з номерами від 1 до n (останнього), на другому від 1 до (n-1) і т.д.

Наведемо приклад сортування методом обміну масиву восьми цілих чисел . Як і в попередньому прикладі для демонстрації на екран виводяться вихідний масив, масив після кожного проходу та результуючий масив.

Приклад:

```
uses crt;
const
   n=8;
var i,j,k,d:integer;
    tmp:integer;
    a : array [1..n] of integer;
begin
    ClrScr;
```

```
    randomize;
    for i:= 1 to n do
    begin
       a[i]:=random(256);
       write (' ',a[i]);
    end;
    writeln;
    writeln;
    d:=n-1;
    for j:=1 to n-1 do
    begin
      for i:=1 to d do
       if ( a[i]>a[i+1]) then
        begin
          tmp:=a[i];
          a[i]:=a[i+1];
          a[i+1]:=tmp;
        end;
       d:=d-1;
       for k:=1 to n do
       write(' ',a[k]);
       writeln;
    end;
    writeln;
    for i:=1 to n do
       write(' ',a[i]);
    readkey;
end.
```

Продемонстровані методи можуть бути модифіковані для збільшення їх ефективності, але викладення цього матеріалу виходить за рамки даного посібника. Ті ж кого зацікавило це питання можуть вивчити його самостійно [3].

Пошук – це процес знаходження серед елементів даного типу елемента з заданими властивостями. Задане значення критерія пошуку називається ключем пошуку. Це може бути умова рівності елемента заданій величині або інша умова. При подальшому розгляді методів пошуку будемо вважати, що кількість елементів даного типу, в якій провадиться пошук – відома і фіксована, тобто не змінюється в процесі пошуку.

Найпростішим, але не самим оптимальним методом пошуку е прямий лінійний пошук. Цей метод використовується тоді, коли немає ніякої додаткової інформації про групу елементів серед якої провадиться пошук. Метод заключається в послідовному перегляді всіх елементів і перевірці їх на відповідність ключу пошуку. Умовою закінчення пошуку може бути або факт знаходження даного елемента, або той факт, що дану сукупність елементів перевірено повністю і не знайдено елементів, що відповідають критерію пошуку. Розглянемо приклад:

```
uses crt;
const
    arraysize=10;
type
   massiv=array [1..arraysize] of integer;
var
     searchkey,element,x:integer;
     a:massiv;
{функція лінійного пошуку}
function linesearch(a:massiv ;key,sizeofarray:integer):integer;
var
```

```
    n:integer;
begin
    for n:=1 to sizeofarray do
    begin
      if (a[n]=key) then
      begin
        linesearch:=n;
        exit;
      end;
    end;
    linesearch:=-1;
end;
{основна програма}
begin
    clrscr;
    {сформуємо випадковим чином масив з кількістю елементів, що дорівнює величині arraysize  і виведемо його на екран}
    randomize;
    for x:=1 to arraysize do
    begin
      a[x]:=random(256);
      write(a[x],' ');
    end;
    writeln;
{отримаємо з клавіатури ключ пошуку}
    writeln('input searchkey');
    readln(searchkey);
{виконаємо пошук елемента масиву, що відповідає заданому ключу}
    element:=linesearch(a,searchkey,arraysize);
```

```
{виведемо  на екран результат пошуку}
if (element<>(-1)) then writeln('found in a[',element,']') {елемент знайдено}
else  writeln('not found'); {елемент не знайдено}
readkey;
end.
```

Такий спосіб пошуку вимагає великих затрат машинного часу. А чи можна якось прискорити пошук потрібного елемента? Очевидно, що без додаткової інформації про задану сукупність елементів, це неможливо. Проте пошук можна зробити значно ефективнішим, якщо відомо, що задана послідовність елементів є впорядкованою за критерієм пошуку. Прикладом такої впорядкованої послідовності може бути телефонний довідник, всі записи якого впорядковано відповідно до абетки. Основною ідеєю пошуку у такій послідовності є вибір деякого випадкового елемента і порівняння його з критерієм пошуку. При цьому може виникнути три випадки:

- елемент відповідає критерію пошуку. Тоді шуканий елемент знайдено і пошук можна завершити;
- елемент має значення більше за величину ключа пошуку. Тоді треба продовжити пошук у тій частині сукупності де значення менші за значення обраного елемента;
- елемент має значення менше за величину ключа пошуку. Тоді треба продовжити пошук у тій частині сукупності де значення більші за значення обраного елемента.

При такій організації пошуку критерієм зупинки може бути або факт знаходження даного елемента, або той факт, що дану сукупність елементів перевірено повністю і не знайдено елементів, що відповідають критерію пошуку. Найпростішим способом реалізації такого алгоритму є метод поділу пополам. При такому методі розглядувану послідовність ділять пополам і порівнюють критерій пошуку з центральним елементом послідовності. Якщо критерій співпадає, то елемент знайдено, якщо значення елементу менше за

заданий критерій то ділять пополам ту частину послідовності де значення елементів більше за значення обраного елемента, якщо ж воно більше, то ділять ту половину де значення елементів менше за значення обраного елемента. Ці дії виконують доти, доки не буде знайдено потрібний елемент, або поки у досліджуваній частині сукупності не залишиться лише один елемен. Оскільки при цьому кожен раз кількість досліджуваних елементів зменшується вдвічі, то швидкість пошуку значно зростає порівняно з лінійним пошуком. Розглянемо приклад:

```
uses crt;
const
    arraysize=15;
type massiv=array[1..arraysize] of integer;
var
  i,key,result:integer;
  a:massiv;
{функція бінарного пошуку}
function binarysearch(b:massiv;searchkey,low,high,size:integer):integer;
var
  middle:integer;
begin
    while(low<=high) do
    begin
    {обираємо середній елемент}
    middle:=round((low+high)/2);
    if (searchkey=b[middle]) then
      begin
        binarysearch:=middle;{елемент знайдено}
        exit;
      end
```

```
        else
          begin
            if(searchkey<b[middle]) then high:=middle-1 {значення ключа більше за значення середнього елемента}
            else                    low:=middle+1;{значення ключа менше}
          end;
        end;
        binarysearch:=-1;{перевірено весь масив елемент не знайдено}
    end;

    {основна програма}
    begin
      clrscr;
      {створимо масив упорядкований за зростанням}
      for i:=1 to arraysize do
      begin
        a[i]:=sqr(i);
        write(a[i],' ');
      end;
      {введемо ключ пошуку}
      writeln('input search key');
      readln(key);
      {виконаємо пошук}
      result:=binarysearch(a,key,1,arraysize,arraysize);
      {виведемо на екран результат пошуку}
      if(result<>(-1)) then writeln('found in a[',result,']'){елемент знайдено}
      else writeln('not found');{елемент не знайдено}
      readkey;
    end.
```

Методи пошуку можна зробити ще ефективнішими, але ми не будемо зупинятись на докладному розгляді цього питання, а відішлемо читача до першоджерел [3].

## 2.2 Строковий тип

Строковий тип (тип string) використовується для обробки текстової інформації. Він подібний до масиву символів, але з тією відмінністю, що елементи такого масива завжди нумеруються з номера 0. При цьому в елементі з номером 0 зберігається величина, що вказує, яка кількість символів записана у дану строку. Самі символи знаходяться у елементах строки починаючи з елемента з номером 1. Звертатись до змінної можна як до цілої строки або поелементно, як до елементів масиву (з урахування вищевказаної особливості).Опис змінної строкового типу має вигляд:

<змінна >:string[довжина строки];

Довжина строки – це число, що вказує, яку максимальну кількість символів можна буде зберігати у вказаній змінній. Це число не може перевищувати 255. При описі змінної строкового типу довжину строки можна не вказувати. В такому випадку довжина строки буде 255 символів. Як уже відмічалось довжина строки вказує яку максимальну кількість символів можна буде зберігати у заданій змінній. Якщо спробувати записати більшу кількість символів, то всі зайві символи буде відкинуто без всякого попередження, тому програміст повинен сам контролювати у яку змінну строкового типу скільки елементів можна записати. Розглянемо приклад:

```
…
var
  st, st1:string[10];{описано строку, що може містити до 10 символів}
  st2:string;{дана строка може містити до 255 символів}
  st3:string[1];{ дана строка може містити лише один символ}
…
```

```
begin
…
  st1:=’1234567890’;{дана строка заповнена повністю}
  st2:=’I am a string’;{в дану строку можна додавати символи}
  st3:=’123’;{в дану строку записано лише символ ‘1’ інші символи відкинуто}
  st1[5]:=’d’; {тепер до st1 записано послідовність символів '1234d67890'}
  st:=st1;{до st записано те саме, що і до  st1}
  st[5]:=st2[1];{ тепер до st записано послідовність символів '1234I67890'}
end.
```

При роботі із строками програміст повинен завжди пам’ятати, що змінна строкового типу займає в пам’яті (довжина строки+1) байт, оскільки існує елемент з номером 0, в якому міститься інформація про кількість символів реально записаних до цієї змінної.

Для полегшення роботи із даними строкового типу існує набір стандартних процедур та функцій. Розглянемо основні з них:

concat(s1 [,s2,…,sn]) – функція типу string, що повертає строкову змінну, яка містить у собі послідовно строки s1…sn;

copy(st, index, count) – функція типу string, що повертає строкову змінну, яка містить count послідовних символів строки st, починаючи з елемента з номером index;

delete(st, index, count) – процедура, що видаляє count елементів із строки st, починаючи з елемента з номером index;

insert(subst, st, index) – процедура, що вставляє послідовність символів, які містяться у змінній subst до строки st, починаючи з позиції з номером index;

length(st) – функція, що вказує кількість символів, що містяться в строковій змінній st;

pos(subst, st) – функція типу integer, що повертає номер позиції строки st, з якої починається послідовність символів, які містяться у змінній subst, або 0, якщо у змінній st нема такої послідовності символів;

str(x[:width[:decimals]],st) – процедура, що перетворює число x будь-якого цілого або дійсного типу у строку символів st; параметри width та decimals, якщо вони присутні, задають формат перетворення;

val(st, x, code) – процедура, що перетворює строку символів st у число, якщо це можливо, і записує його у змінну x. До змінної code записується 0, якщо перетворення успішне, або номер позиції першого символа змінної st, який не може бути перетворений.

У даному переліку між символами [ ] знаходяться необов'язкові аргументи. Наведений список не є повним. Більш повну інформацію можна одержати із файлів допомоги системи програмування Паскаль. А ми наведемо приклад використання описаних процедур та функцій:

```
var
  x:real;
  y:integer;
  st, st1:string;
begin
  st:=concat('12','345'); {st:='12345'}
  st1:=copy(st, 3, length(st)-2); {st1:='345'}
  insert('-', st1, 2); {st1:='3-45'}
 delete(st,pos('2',st),3); {st:='15'}
 str(pi:6:2,st); {st:='  3.14'}
end.
```

Над строками можна виконувати операції відношення =,<>,>,<,>=,<=. Ці операції виконуються посимвольно. Зліва направо порівнюються коди символів. Якщо одна строка містить менше символів ніж друга то система на час порівняння доповнює її справа символами з кодом 0.

## 2.3 Тип множина

Множина – це кінцевий набор значень деякого базового типу. Базовий тип не може мати більше 256 можливих значень, і порядкові значення нижньої та верхньої межі не повинні виходити за межі діапазона 0..255. В якості базового може бути використаний любий порядковий тип, крім типів word, integer, LongInt. Будь-яка множина може приймати значення [], що називається порожньою множиною.

Опис множини має вигляд:

<ім’я множини>=set of <базовий тип>;

Наприклад:

```
type
    digitChar=set of ‘0’..’9’;{тип множина цифрових символів}
    digit=set of 0..9;{тип множина цілих чисел від 0 до 9}
var
    s1,s2,s3:digitchar;
    s4,s5,s6:digit;
```

Над множинами можна виконувати такі операції:

“+” – об’єднання двох множин з сумісними базовими типами ;

“-” – різниця двох множин з сумісними базовими типами;

“*” – перетин двох множин з сумісними базовими типами.

Результати операцій над множинами підпорядковуються правилам математичної логіки множин, а саме:

- порядкове значення С належить множині (А+В) тільки в тому випадку, коли воно належить множині А або множині В;
- порядкове значення С належить множині (А-В) тільки в тому випадку, коли воно належить множині А, але не належить множині В;

- порядкове значення С належить множині (A*B) тільки в тому випадку, коли воно належить і множині А, і множині В одночасно;

Допустимою також являється операція перевірки належності елемента до заданої множини. Перевірка здійснюється за допомогою оператора “in”. Цей оператор повертає логічне значення “true”, якщо даний елемент належить заданій множині і значення “false”, якщо не належить.

Крім того існує дві процедури, які полегшують процесс формування множин – це процедура Include(var s: set of T; I:T), яка додає елемент до множини, та процедура Exclude(var s: set of T; I:T), яка видаляє вказаний елемент із множини.

Наприклад:

```
type
   digitChar = set of ‘0’ .. ‘9’;
   digit       = set of 0 .. 9;
var
   s1,s2,s3:digitchar;
   s4,s5,s6:digit;
begin
   s1 :=[‘1’,’2’,’3’]; { задаємо множину s1, що складається з символів 1, 2, 3}
   s2 :=[‘3’,’4’,’5’]; { задаємо множину s2, що складається з символів 3, 4, 5}
   s3 :=s1+s2; {множина s3 складатиметься із символів 1, 2, 3, 4, 5}
   s4 :=[0..3,6]; { множина s4 складатиметься із цифр 0, 1, 2, 3, 6}
   s5 := [2,3]; { множина s5 складатиметься із цифр 2, 3}
   s6 := s4-s5; { множина s6 складатиметься із цифр 0,1,6}
   s6:=s4*s5; { множина s6 складатиметься із цифр 2,3}
   if (1 in s6 ) then writeln(‘Yes’)  {перевіряємо чи належить цифра 1 до s6}
```

```
else writeln('No');        {і друкуємо Yes, якщо належить, або No, якщо не
                            належить. В даному випадку буде надруковано
                            No}
include(s6,4);  {множина s6 складатиметься з елементів 2,3,4}
exclude(s6,2);  { множина s6 складатиметься з елементів 3,4}
end.
```

Окрім перелічених вище операцій над множинами допустимою є також операція порівняння двох множин. При цьому діють такі правила:

- вираз A=B істинний тільки тоді, коли A і B містять однакові елементи;
- вираз A<>B істинний, якщо хоч одна із заданих множин містить хоч один елемент, відсутній в іншій;
- вираз A<=B істинний, якщо кожен елемент множини A являється також елементом множини B;
- вираз A>=B істинний, якщо кожен елемент множини B являється також елементом множини A.

Використання множин дозволяє зробити програму більш ефективною за рахунок зменшення кількості перевірок.

## 2.4 Тип запис

Запис – це структура даних, що складається з фіксованого числа компонентів. Ці компоненти називають полями запису. Поля запису можуть бути однакових чи різних типів. Щоб можна було звертатись до конкретного поля даного запису, кожному полю присвоюється своє власне ім'я.

Структура опису типу запис така:

<ім'я типу>=record<список полів>end.

Тут <ім'я типу> - ідентифікатор створюваного типу даних;

record, end – зарезервовані слова;

<список полів> - список імен полів з указанням їх типів, між якими ставиться крапка з комою.

Наприклад:

```
type
  personal_data=record
  day, month:byte;
  year:word
end;
…
var
  a,b:personal_data;
```

В даному прикладі об’явлено новий тип даних personal_data, що є записом. Даний запис включає три поля, два з яких (day та month) є змінними типу byte, а третє (year) – змінна типу word.

Значення змінних типу запис можна передавати іншим змінним того ж типу, наприклад

```
a:=b;
```

Можна отримати доступ до кожного з полів запису, якщо використовувати комбіноване ім’я. Комбіноване ім’я складається з двох частин – імені запису та імені поля розділених крапкою:

```
a.day:=10;
b.year:=1967;
```

Полем запису може бути не лише проста змінна, але і змінна складного типу – масив, інший запис змінна строкового типу і т.д. В цьому випадку комбіноване ім’я може бути більш складним:

```
type
  personal_data=record
  day, month:byte;
  year:word
```

```
end;

  anket_data=record
  own_data:personal_data;
  qualification: string[20] ;
end;
…
var
  c:anket_data;
…
c.own_data.day:=5;
c.qualification:=’студент’;
```

Для полегшення доступу до полів запису можна використовувати оператор приєднання with:

```
…
var
  a,b:personal_data;
…
with a do
begin
day:=1;
year:=1967;
month:=12;
end;
with b do
begin
day:=10;
year:=1977;
month:=11;
```

```
end;
```

Використовуючи раніше об’явлені типи-записи можна формувати більш складні типи даних, такі як,наприклад, масиви записів:

```
…
var
  d:array[1..5] of personal_data;
…
d[1].year:=10;
…
d[2].day:=2;
```

Такий тип даних зручно використовувати для автоматизації обробки великої кількості однотипних записів, коли наперед відома їх кількість.

Але при програмуванні часто зустрічаються ситуації, коли наперед невідомо яку кількість записів доведеться обробляти. Така ситуація можлива, наприклад, при обробці бази даних. Тоді для автоматизації процесу обробки записів використовують більш складний тип даних –список. Цей тип даних настільки часто використовується, що для його розгляду виділимо цілий підрозділ.

## 2.5 Списки

Список – це сукупність записів, кількість яких може змінюватись в ході виконання програми. При цьому пам’ять для розміщення нового запису виділяється і вивільняється динамічно по мірі необхідності. Зрозуміло, що для реалізації такого типу даних необхідно використовувати покажчики, тому основним елементом такого списку являється запис, що має одне або кілька полів, які є покажчиками на його власний тип:

```
type
   dinptr=^dinrec;
   dinrec=record
```

```
  re:real;
  next:dinptr;
  end;
```

В даному прикладі створено тип-запис dinrec. Зверніть увагу на поле next типу dinrec, яке є покажчиком на тип dinrec. Наявність таких полів у записах дає можливість зв’язувати окремі записи у списки.

Для ефективного використання списків необхідно навчитися виконувати над списками операції створення списків, заповнення даними записів списку, додавання запису до списку, вилучення запису із списку. Розглянемо ці операції на прикладі:

```
uses crt;
type
  dinptr=^dinrec;{опишемо тип-покажчик на запис потрібної структури}
  dinrec=record    {опишемо тип-запис потрібної структури}
  re:real;
  next:dinptr;
  end;

var
  first:dinptr;        {об’явимо покажчик, в якому будемо зберігати адресу першого елементу списку}
  elem,tmp:dinptr;    {об’явимо ще два покажчики. Вони знадобляться при виконанні операцій додавання та знищення елементів }
  i,counter:integer;  {об’явимо змінні цілого типу. Змінна і буде використовуватись для організації циклів, counter для збереження числа записів наявних у списку}

begin
 clrscr;
```

```
  counter:=0;          {початкова кількість записів 0}
  first:=nil;
  new(elem);           {виділяємо пам’ять під перший запис}
  elem^.next:=nil;     {присвоюємо початкові значення}
  tmp:=nil;
    first:=elem; {зберігаємо адресу початку списку}
{створимо і додамо до списку 5 записів, збільшуючи щоразу на одиницю змінну counter}
  for i:=1 to 5 do
  begin
     elem^.re:=sqr(i); {записуємо дані в інформаційне поле запису}
     new(elem^.next); {виділяємо пам’ять під наступний запис}
     elem:=elem^.next; {переміщуємось до кінця списку}
     inc(counter);
  end;

    tmp:=first;       {tmp тепер вказує на початок списку, як і first}
  {виведемо список на екран}
  for i:=1 to counter do
  begin
      writeln(tmp^.re:7:2); {виводимо на екран інформаційне поле запису}
      tmp:=tmp^.next; {переміщуємось до наступного запису}
  end;
  readkey;
  tmp:=first; {перейдемо в початок списку}
  {знищимо 3-ій запис}
  for i:=1 to 2 do {переходимо до 3-го запису}
  begin
    elem:=tmp;      {змінна містить 2-го запису}
```

```
   tmp:=tmp^.next; {змінна містить адресу 3-го запису}
  end;
  elem^.next:=tmp^.next;{зв'яжемо 2-ий елемент з четвертим, відкинувши
  третій елемент}
  dec(counter); {зменшимо на одиницю counter }
  dispose(tmp); {звільнимо пам'ять, що містила 3-ій запис}
  writeln;
  {виведемо на екран список після  видалення 3-го запису}
  tmp:=first;
  for i:=1 to counter do
  begin
      writeln(tmp^.re:7:2);
      tmp:=tmp^.next;
  end;
  readkey;
{знищимо 1-ий запис}
  tmp:=first;
  first:=first^.next;
  dec(counter);{зменшимо counter}
  dispose(tmp);{звільнимо пам'ять, що містила 1-ий запис}
{виведемо список на екран}
 tmp:=first;
  writeln;
  for i:=1 to counter do
  begin
      writeln(tmp^.re:7:2);
      tmp:=tmp^.next;
  end;
  readkey;
```

```
 {вставимо новий запис у початок списку}
 tmp:=first;
 new(elem);
 elem^.re:=pi; {запишемо у інформаційне поле запису число 3.14}
 elem^.next:=first;
 first:=elem;
 inc(counter);{збільшимо counter на 1}
{виведемо список на екран}
tmp:=first;
 writeln;
 for i:=1 to counter do
 begin
     writeln(tmp^.re:7:2);
     tmp:=tmp^.next;
 end;
 readkey;
{вставимо запис у 3-тю позицію}
 tmp:=first;
 for i:=1 to 2 do {перейдемо до того місця списку, куди треба вставити запис }
 begin
     elem:=tmp; {збережемо адресу 2-го запису}
     tmp:=tmp^.next; {збережемо адресу 3-го запису}
 end;
 new(elem^.next); {виділимо пам'ять під запис}
 elem^.next^.re:=49; {запишемо до інформаційного поля запису число 49}
 elem^.next^.next:=tmp; {вставимо запис до списку}
 inc(counter); {збільшимо counter на 1 }
 writeln;
```

```
{виведемо список на екран}
 tmp:=first;
 for i:=1 to counter do
 begin
    writeln(tmp^.re:7:2);
    tmp:=tmp^.next;
 end;
 readkey;
 {перейдемо в початок списку}
 tmp:=first;
{послідовно знищимо весь список. Спочатку 1-ий запис, потім 2-ий і т.д.}
 for i:=1 to counter do
 begin
    elem:=tmp    ; {збережемо адресу поточного запису }
    tmp:=tmp^.next ;{перейдемо до наступного запису}
    dispose(elem); {знищимо запис, чию адресу збережено на
попередньому кроці і вивільнимо пам'ять зайняту ним}
    dec(counter); {зменшимо counter на 1}
 end;
 first:=nil;
 tmp:=first;
end.
```

З прикладу видно, що навіть елементарні операції із списками потребують досить високої кваліфікації, але вони дають змогу оптимально використовувати пам'ять. В наведеному прикладі розглянуто всі базові операції необхідні для роботи із списками. Приклад добре прокоментовано у тексті програми і додаткових пояснень не потребує. Тим же хто хоче розібратись із цим питанням більш досконало рекомендуємо звернутись до літератури [4].

## 2.6 Файлові типи

### 2.6.1 Загальні положення

Файл у Паскалі – це або поіменована область зовнішньої пам’яті ПК(диску, дискети , “віртуального” диску), або пристрій – носій інформації.

Будь-який файл має три особливості:

- він має ім’я, що дає змогу програмі працювати з кількома файлами одночасно;
- він містить дані одного типу;
- під час створення файлу невідомо, яку він матиме довжину при подальшому використанні.

В Паскалі можна створити файли, що містять дані будь-якого з допустимих типів, крім файлового типу. Тобто не можна створити файл файлів. Всі файли, які використовуються в Паскалі, можна розділити на три групи :

- типізовані файли;
- текстові файли;
- нетипізовані файли.

Типізовані файли – це файли, які містять дані певного типу. Текстові файли – це файли, що містять текстову інформацію. Ці файли слід відрізняти від типізованих файлів, що містять символи чи строки. Нетипізовані файли – це файли, до яких інформація записується без урахування типу. Фактично – це канали вводу-виводу інформації нижнього рівня, що можуть використовуватись для прямого доступу до будь-якого файлу без урахування його типу та структури.

Змінна файлового типу – це змінна, що містить інформацію, про місцезнаходження та тип файлу. Розглянемо приклад:

```
type
  product = record
        name:string;
```

```
        code:word;
        cost:real;
end;
text80 = file of string[80];
…
var
  f1: file of char; {змінна типу файл символів. Файл, на який вказує f1 містить символи}
  f2: text; {змінна типу текстовий файл. Файл f2 містить інформацію у вигляді тексту}
  f3: file; {змінна-нетипізований файл. Структура файлу не має значення. Читання інформації з файлу та запис інформації у файл здійснюватиметься побайтно}
  f4:text80; {змінна типу text80, описаного у розділі type. Інформація у файлі зберігається у вигляді строк по 80 символів}
  f5:file of product; { змінна типу product, описаного у розділі type. Інформація у файлі структурована у вигляді записів типу product }
```

### 2.6.2 Порядок роботи з файлами в Паскалі. Типізовані файли

Як уже вказувалось змінна файлового типу містить інформацію про структуру файлу та його місцезнаходження. Структура файлу визначається типом файлової змінної (дивись попередній приклад), а для визначення місцеположення файлу необхідно зв’язати змінну з конкретним файлом за допомогою процедури assign. Коли зв’язок файлової змінної із зовнішнім файлом встановлено, то для підготовки файлу до виконання операцій вводу-виводу його необхідно відкрити. Файл, що вже існує можна відкрити за допомогою процедури reset, а новий файл можна створити і відкрити за допомогою процедури rewrite. При цьому завжди слід пам’ятати, що текстові файли, відкриті за допомогою процедури reset доступні лише для читання, а відкриті за допомогою процедури rewrite доступні лише для запису. Крім того

при роботі з текстовими файлами можна використовувати процедуру append. Ця процедура дозволяє відкрити текстовий файл в режимі доповнення (запису інформації в кінець файлу).

Типізовані і не типізовані файли дозволяють роботу як в режимі читання, так і в режимі запису незалежно від того, якою процедурою вони були відкриті (rewrite чи reset).

Коли запускається будь-яка програма, то завжди автоматично відкриваються два стандартних файли input та output (ввід та вивід). Зазвичай input – це файл зв'язаний з клавіатурою. Він доступний лише для читання. Файл output зазвичай зв'язаний з монітором. Цей файл доступний лише для запису.

Доступ до інформації, що міститься в файлі, організується, як правило, послідовно за допомогою стандартних процедур read та write. Однак при роботі з типізованими і не типізованими файлами можна організувати прямий доступ до заданого елементу файлу за допомогою процедури seek. Поточну позицію в файлі можна визначити з допомогою стандартної функції filepos, а розмір файлу за допомогою стандартної функції filesize.

Після того, як програма завершить роботу з файлом, його необхідно закрити за допомогою процедури close. Після цього файлова змінна може бути зв'язана з іншим файлом.

Нижче наведено основні процедури та функції для роботи з файлами:

assign(var f; name:string) – процедура, що зв'язує файлову змінну f з конкретним файлом, шлях до якого задано у строковій змінній name;

chdir(s:string) – процедура, що робить поточним каталог, ім'я якого задане у строковій змінній s;

close(var f) – процедура, що закриває файл зв'язаний з файловою змінною f;

erase(var f) – процедура, яка знищує файл зв'язаний з файловою змінною f;

getdir(d:byte; var s:string) – записує у строку s назву поточного каталогу на вказаному диску d;

mkdir(s:string) – процедура, що створює каталог з назвою заданою у строковій змінній s;

rename(var f; newname:string) – процедура, що змінює ім’я файла, зв’язаного з файловою змінною f, на ім’я задане у строковій змінній newname;

reset(var f) – процедура, яка відкриває існуючий файл, зв’язаний з файловою змінною f;

rewrite(var f) – процедура, що створює новий файл, зв’язаний з файловою змінною f;

rmdir(s:string) – дана процедура знищує каталог, ім’я якого задане у строковій змінній s, якщо він не містить файлів;

seek(var f; n: longint) – ця процедура переміщує покажчик поточної позиції файлу, зв’язаного з файловою змінною f, у позицію з номером n. Дана процедура не може використовуватись при роботі з текстовими файлами;

truncate(var f) – процедура, що усікає файл, зв’язаний з файловою змінною f до поточної позиції. Вся інформація, яка знаходилась у файлі після цієї позиції знищується;

eof(var f) – функція, що повертає значення “true”, якщо досягнуто кінець файлу, зв’язаного з файловою змінною f або “false” у противному випадку;

filepos(var f) – функція, яка повертає поточну позицію у файлі, зв’язаному з файловою змінною f. Функція не може бути використана при роботі з текстовими файлами;

filesize(var f) – функція, що повертає поточний розмір файлу, зв’язаного з файловою змінною f. Функція не може бути використана при роботі з текстовими файлами;

ioresult – функція, яка повертає ціле значення, що характеризує стан останньої операції вводу-виводу.

Зазвичай при всіх зверненнях до стандартних функцій та процедур вводу-виводу автоматично виконується перевірка на наявність помилок. При наявності помилки програма завершує роботу і виводить на дисплей

повідомлення про помилку. За допомогою директив компілятора {$I+} та {$I-} цю автоматичну перевірку можна включити чи виключити. Коли автоматична перевірка виключена (директива {$I-}), помилки вводу-виводу, що виникають при роботі програми, не приводять до її зупинки. При цьому можна виконати перевірку результату операції за допомогою функції ioresult і запрограмувати потрібну реакцію програми на помилку. Наприклад:

```
uses crt;
var
  f : file of string; {файлова змінна типу файл строк}
      buf,buf1: string;
begin
      clrscr;
  assign(f,'myfile.dat'); {зв'язуємо файлову змінну f з файлом myfile.dat,
  який знаходиться в поточному каталозі}
  {$I-} {відключаємо автоматичний контроль помилок вводу-виводу}
  reset(f); {відкриваємо файл myfile.dat}
  {$I+} {поновлюємо режим автоматичного контролю помилок вводу-
  виводу}
  {аналізуємо стан виконання команди reset}
  if (ioresult<>0) then
  {якщо виникла помилка (тобто такого файлу немає) виводимо на екран
  повідомлення про помилку і створюємо потрібний файл}
        begin
          writeln('File not found, creating new file'); {}
          rewrite(f) ;
        end
  else
  {в противному випадку виводимо на екран повідомлення про успішне
  відкриття файлу}
```

```
    begin
      writeln('File opening successfuly!');
    end;
     seek(f,filesize(f)); {переводимо покажчик поточної позиції файлу у
     кінець файлу}
  buf:='Hello, World !';
  writeln('Writing to file: ',buf); { повідомляємо про запис до файлу}
     write(f,buf); {записуємо у кінець файлу строку buf}
  writeln;
  writeln('Reading from file: '); {повідомляємо про читання інформації з
  файлу}
  seek(f,0); {переходимо у початок файлу}
  while( not eof(f)) do {поки не досягнемо кінця файлу виконуємо
  вказані нижче дії}
  begin
       read(f,buf1); {читаємо інформацію з файлу построково}
       writeln(buf1); {виводимо прочитану інформацію на екран}
  end;
  close(f); {закриваємо файл myfile.dat}
  readkey;
end.
```

### 2.6.3 Робота з текстовими файлами

Текстові файли зв’язуються із файловими змінними стандартного типу text. Текстові файли призначені для збереження текстової інформації. При роботі з ними слід пам’ятати, що в Паскалі тип text відрізняється від типу файл символів (file of char) та від типу файл строк (file of string).

Текстовий файл інтерпретується як сукупність строк змінної довжини. Кожна строка закінчується спеціальним символом, який так і називається – символ кінця строки. Доступ до кожної строки здійснюється послідовно,

починаючи з першої. Для запису до файлу використовують процедури write та writeln. Для читання з файлу використовують процедури read та readln(див. нижче). Як уже було сказано доступ до файлу здійснюється послідовн. Так нові строки додаються завжди у кінець файлу, а читання файлу після відкриття починається з першої строки.

Для роботи з текстовими файлами в Паскалі існують наведені нижче процедури та функції (квадратними скобками позначено необов'язкові параметри):

append(var f: text) – процедура, що відкриває текстовий файл для додавання інформації;

flush (var f:text) – процедура, яка при використанні буферизованого вводу-виводу змушує систему записати інформацію із буферу, вводу-виводу, пов'язаного із файловою змінною f, безпосередньо у файл зв'язаний із цією змінною ;

read([var f: text;] v1[,v2…]) – зчитує одне чи кілька значень із файлу зв'язаного із файловою змінною f. Якщо файлова змінна не вказана то дані зчитуються із стандартного файлу вводу – input;

readln ([var f: text;] v1[,v2…]) – процедура аналогічна попередній, яка додатково після зчитування переходить до початку навої строки;

settextbuf( var f: text; var buf [; size:word]) – процедура, що назначає буфер вводу-виводу для текстового файлу;

write([var f: text;] v1[,v2…]) – записує одне чи кілька значень до файлу зв'язаного із файловою змінною f. Якщо файлова змінна не вказана то дані записуються до стандартного файлу виводу – output;

writeln([var f: text;] v1[,v2…]) – процедура виконує дії аналогічні попередній, а потім додає до тексту символ кінця строки;

eoln(var f: text) – функція, що повертає "true", коли досягнуто кінець строки, та "false" у противному випадку;

seekeoln(var f: text) – функція аналогічна попередній;

seekeof(var f: text) – функція, що повертає “true”, коли досягнуто кінець файлу, та “false” у противному випадку.

Розглянемо приклад :

```
uses Crt;
var
  f : Text;
   j : Integer;
begin
  Assign(f,'TEST.TXT');
  Rewrite(f);
  { створимо файл і запишемо до нього  2-і строки, кожна з яких містить
  чотири цифрових символи і кілька символів пробілу }
  Writeln(f,'1 2 3 4 ');
  Writeln(f,'5 6 7 8 ');
  Reset(f);
  { читаємо записане до файлу, SeekEoln повертає TRUE якщо в поточній
  стрічці більше нема цифрових символів; SeekEof повертає TRUE, якщо
  далі у файлі немає більше інших символів крім символів пробілу }
  while not SeekEof(f) do
  begin
    if SeekEoln(f) then
      Readln; { переходимо до іншої строки }
    Read(f,j);
    Writeln(j);
  end;
end.
```

### 2.6.4 Робота з нетипізованими файлами

Нетипізовані файли відрізняються тим, що для їх компонентів не вказано тип. Відсутність типу робить ці файли, з одного боку, сумісними з любими іншими файлами, а з іншого – дозволяє організувати швидкий обмін даними між пам’яттю та диском.

При ініціалізації нетипізованих файлів процедурами reset чи rewrite можна вказати довжину запису в байтах. Наприклад:

```
var
  f:file;
…
begin
…
  reset(f,512);
…
end.
```

В наведеному прикладі файл, пов’язаний із нетипізованою файловою змінною f, буде використовуватися в програмі, як нетипізований файл з довжиною запису 512 байт. Якщо при ініціалізації файлу довжину опущено, то по замовчуванню вважається, що довжина рівна 128 байтам. В Паскалі довжина запису повинна бути додатньою і не перевищувати 65535 байт. Для прискорення роботи з файлом рекомендується вибирати довжину запису кратною величині сектора дискового носія інформації (величина одного сектора – 512 байт). Максимальної швидкості при роботі з нетипізованим файлом можна досягти при довжині запису рівній довжині одного кластера. Величина кластера може бути різною у різних обчислювальних системах (два сектори і більше).

При роботі з нетипізованими файлами можуть бути використані всі процедури і функції, що використовуються при роботі з типізованими файлами,

крім read та write. Замість них при роботі з нетипізованими файлами використовуються процедури blockread та blockwrite:

blockread(<файлова змінна>,<буфер>,<кількість записів> [,<результат>]) – процедура призначена для зчитування інформації з нетипізованого файлу.

blockwrite(<файлова змінна>,<буфер>,<кількість записів> [,<результат>]) – процедура призначена для запису інформації до нетипізованого файлу.

Параметр <файлова змінна> вказує, над яким файлом буде виконано операцію вводу-виводу. Параметр <буфер> вказує ім’я змінної, у яку буде записуватись прочитана інформація при зчитуванні або із якої змінної буде записано інформацію у файл при запису інформації. Параметр <кількість записів> вказує, яку кількість записів треба прочитати чи записати. Змінна, у яку буде записуватись зчитана інформація, повинна мати розмір достатній для збереження заданої кількості записів. Необов’язковий параметр <результат> після виконання заданої процедури містить кількість записів, які реально вдалося прочитати з файлу або записати у файл. Він використовується для аналізу помилок при роботі з файлом і організації поведінки програми при виникненні таких помилок.

Розглянемо приклад :

```
uses Crt;
var
  FromF, ToF: file;
  NumRead, NumWritten: Word;
  Buf: array[1..2048] of Char;
begin
  clrscr;
  Assign(FromF, 'c:\myin.txt'); { відкриваємо вхідний файл }
  Reset(FromF, 1); {розмір запису 1 байт}
  Assign(ToF, 'c:\myout.txt'); { відкриваємо вихідний файл }
  Rewrite(ToF, 1);  { розмір запису 1 байт }
```

```
repeat
  BlockRead(FromF, Buf, SizeOf(Buf), NumRead); {читаємо з вхідного файлу}
  BlockWrite(ToF, Buf, NumRead, NumWritten); {записуємо увихідний файл }
 until (NumRead = 0) or (NumWritten <> NumRead); {повторюємо ці дії доки не перепишемо весь вхідний файл}
 Close(FromF);
 Close(ToF);
end.
```

## Контрольні запитання

1. Що таке масив?
2. Як описуються масиви у програмі на алгоритмічній мові Паскаль?
3. Що таке сортування? Для чого воно потрібне?
4. Які є методи сортування?
5. В чому полягає метод сортування за допомогою включення?
6. Як виконується сортування за допомогою вибору?
7. Який алгоритм сортування за допомогою обміну?
8. Порівняйте ці три методи?
9. Що таке пошук?
10. Які є методи пошуку?
11. Який алгоритм методу прямого лінійного пошуку?
12. Коли доцільно використовувати метод прямого лінійного пошуку?
13. Який алгоритм пошуку методом поділу навпіл?
14. Що таке строковий тип?
15. Як описати змінну строкового типу?
16. Які є процедури і функції для роботи із строковим типом у алгоритмічній мові Паскаль?
17. Що таке множина?
18. Як описати множину у програмі на алгоритмічній мові Паскаль?
19. Які операції можна виконувати над множинами у Паскалі?
20. Як перевірити чи належить елемент до множини?
21. Які є процедури для роботи з множинами?
22. Як порівняти дві множини?

23. Що таке запис?
24. Що таке поле запису?
25. Для чого потрібний тип запис?
26. Як описати змінну типу запис у алгоритмічній мові Паскаль?
27. Як звернутись до конкретного поля типу запис?
28. Що таке масив записів?
29. Як звернутись до конкретного поля заданого елемента масиву записів?
30. Що таке список?
31. Як створити список?
32. Як звернутись до конкретного елементу списку?
33. Що таке файл?
34. Які типи даних може містити файл?
35. На які групи можна розділити файли у Паскалі?
36. Що таке типізований файл?
37. Що таке не типізований файл?
38. В чому різниця між типізованими і нетипізованими файлами?
39. Що представляє собою текстовий фай, чим він відрізняється від файлу символів?
40. Як описати змінну файлового типу у програмі на алгоритмічній мові Паскаль?
41. Які є у Паскалі процедури та функції для роботи з типізованими файлами?
42. Які є у Паскалі процедури та функції для роботи з нетипізованими файлами?
43. Які є у Паскалі процедури та функції для роботи з текстовими файлами?
44. Які особливості роботи з текстовими файлами у Паскалі?
45. Які дії необхідно виконати для відкриття і для закриття файлу?

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Фараонов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие.-М.: «Нолидж», 1997.-616 с.
2. Клименко А.К. Kylix 1.0. Базы данных и приложения. Лекции и упражнения. – К.: Издательство «ДиаСофт», 2001. – 288 с.
3. Вирт Н. Алгоритмы и структуры данных : Пер. с англ.-2-е изд., испр.-СПб: Невский Диалект, 2001.-352 с.
4. Абрамов С.А., Зима Е.В. Начала программирования на языке паскаль. – М.: Наука. Гл. ред. физ.-мат. лит., 1987. – 112 с.