1. Мови програмування. Класифікація мов програмування. Спрощена модель компілятора. Етапи розв’язку задачі на ПЕОМ.

**Мова програмування** — будь-яка система позначень і понять для описань структур даних і алгоритмів.

1) Алфавіт (набір символів, які дозволено використовувати);

2) Синтаксис (система правил, по яких записуються конструкції мови);

3) Семантика (набір правил, за якими тлумачаться ці конструкції).

Найвідоміші мови програмування:

1. Fortran (1954);
2. Cobol (1958);
3. Algol (1960);
4. PL/1 (1963);
5. Basic (1964);
6. Pascal (1968);
7. Turbo Pascal;
8. ADA (1975);
9. C;
10. C ++;
11. C#.

Класифікація і характеристики мов програмування:

1. По реалізованих принципах програмування:

· мови низького рівня (конструкції мови близькі до машинного коду);

· мови високого рівня (конструкції мови близькі до термінів вхідного завдання);

2. По класу вирішуваних задач:

· універсальні;

· спеціальні.

1 . Лексичний аналіз . Серед множини всіх символів є так звані символи-роздільники, які розбивають програми на окремі слова. Ці слова називаються лексемами.

2 . Синтаксичний (граматичний) аналіз. Перевіряє коректність записів згідно до синтаксичних правил мови та переводить посліідовність лексем у внутрішній код компілятора. Внутрішній код компілятора – це дії, які мусить виконати комп’ютер, але це не кінцевий машинний код.

3 . Кінцевий машинний код отримується з генератору коду.

В прооцесі роботи всі блоки звертаються до набору таблиць, в яких зберігається постіна для трансляції усіх програм інформація та інформація для трансляції кожної програми.

Етапи розв’язку задачі на ПЕОМ;

1. Постановка задачі (формалізація);
2. Складання алгоритму (алгоритмізація);
3. Складання програми (програмування);
4. Відладка програми (пошук та виправлення синтаксичних помилок);
5. Тестування (пошук та виправлення семантичних помилок);
6. Документування.

У конкретних реалізаціях компілятора ці етапи можуть бути розділені, або навпаки поєднані в тому чи іншому вигляді.

1. Алфавіт мови Паскаль. Спеціальні символи, зарезервовані слова, ідентифікатори, мітки, коментарі. Структура програми. Заголовок програми. Розділи описань: модулі, мітки, константи, типи, змінні, процедури та функції; розділ операторів. Приклади.

**Алфавіт мови**

Імена у мові Pascal формуються з латинських літер A-Z, a-z, цифр 0-9 та знака «підкреслення» («\_»). Також використовуються спеціальні символи : + — \* / = < > [ ], () ; { } $ ^ # . Будь-яке ім'я повинно починатися з латинської літери або символу «підкреслення» (від англ. «underscore»), виключення становлять лише імена міток. В Паскалі не враховується регістр літер.

Також до символів мови Pascal належать :=, <>, <=, >=, .., (. .), (\* \*).

Всі слова Pascal діляться на

1) зарезервовані слова

2) ідентифікатори користувача

3) стандартні ідентифікатори.

Зарезервовані слова – слова, що мають чітко визначене значення і фіксований смисл. Їх не можна використовувати як ідентифікатори.

Зарезервовані слова:

and, asm, array, begin, case, const, constructor, destructor, div, do, downto, else, end, exports, file, for, function, goto, if, implementation, in, inherited, inline, interface, label, library, 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, absolute, assembler, export, external, far, forward, index, interrupt, near, private, public, resident, virtual.

Використання імен псевдомасивів Port, PortW, Mem, MemW, MemL не викликає помилки, але унеможливлює їх подальше застосування.

Ідентифікатори користувача – імена, що використовують для позначення міток, констнт, змінних, що визначені самим програмістом.

Стандартні ідентифікатори служать для визначених розробниками мови типів, процедур, функцій, констант. Стандартні ідентифікатори, на відміну від зарезервованих слів, можуть бути перевизначені.

Індетифікатор починається тільки з літер латаниці, або символу підкреслювання (вийнятком являються мітки, які можуть починатися з цифр) вони не ПОВИННІ МІСТИТИ НІЧОГО КРІМ СИМВОЛІВ ЛАТИНИЦІ, ЦИФР, та \_, максимальна довжина 127 символів, але розрізняються перші 63.

Коментарі в коді програми оформлюються фігурними дужками {коментар} або ж сполученням звичайних дужок із зірочкою (\*коментар\*).

Коментарі – це пояснення програми, або її окремих частин, наявність коментарів робить програму більш зручною для читання.

Програма мовою Паскаль складається з заголовка і блока, що складється з розділу модулів, розділу міток, розділу констант, розділу типів, розділу змінних, розділу процедур і функцій, розділу операторів.

<програма> ::= <заголовок програми>; <блок>.

<заголовок програми> ::= program <ім’я програми>.

< ім’я програми > ::= <ідентифікатор>.

Приклад: program \_BeautifulDay

Блок – розділи описань програми.

<блок> ::=<розділ модулів>; <розділ міток>; <розділ констант>; <розділ типів>; <розділ змінних>; <розділ процедур/функцій>; <розділ оперторів>.

1. **Розділ модулів** визначається службовим словом USES і містить імена модулів, що підключаються (бібліотек), як тих, що входять до складу системи Turbo Pascal, так і тих, що написані користувачем власноруч. Розділ опису модулів повинний бути першим серед розділів описів. Імена модулів відокремлюються один від одного комами:

<розділ модулів>::= <пусто> | uses <ім’я модуля> {,< ім’я модуля >};

Приклад: uses crt, graph;

2. **Розділ міток.** Будь-який оператор у програмі може бути позначений міткою. Ім'я мітки задається за правилами створення ідентифікаторів Турбо Паскаль. У якості мітки також можуть використовуватися довільні цілі числа без знаку, що містять не більше чотирьох цифр. Мітка ставиться перед оператором і відокремлюється від нього двокрапкою. Усі мітки, що використовуються в програмі, повинні бути перераховані в розділі опису міток.

Приклад: label 10, 100, krok1;

3. **Розділ констант**:

Приклад: const m = 100; l = -m; p = ‘turbo’; k = m \* l.

4. **Розділ типів.** Кожний елемент даних належить до одного з типів, припустимих для даної мови. Тип визначає множину значень даних і сукупність операцій, припустимих до них.

Приклад: type litera = ‘a’..’z’; vidrizok = -10..10.

**5.** **Розділ змінних.**

Приклад: var P,Q,R: Integer; A,B: Char; F1,F2: Boolean;

6. **Розділ процедур та функцій**.

7. **Розділ операторів** являє собою складений оператор, що міститься між службовими словами Begin ……….. End.

1. Типізовані константи: описання та використання. Типізовані константи стандартних типів. Типізовані константи-множини. Типізовані константи-масиви. Типізовані константи-записи. Типізовані константи процедурного типу. Приклади.

Константи бувають:

1. Прості

Приклад: const length = 100; min = -1; bool = true;

1. Типізовані (ініціалізована змінна)

Вони задаються в розділі оголошення констант у такий спосіб:

<ідентифікатор> : <тип> = <значення>

Приклад: const n: integer = 100; ch: char = ‘a’

Типізовані константи-множини

type digits = set of 0..9;

const

D1: didits = [0,2,3]

Типізовані константи-масиви.

Const DigVector: array [1..3] of Real = (0.1, 2.5, 1.2);

Типізовані константи-записи.

type rec = record

r:real;

b:boolean;

end;

const A: rec = (r:3.1; b: true);

Типізовані константи процедурного типу.

Type Ftype = Function (X: Real);

Function Sh(X: Real): real; far;

begin

sh:=exp(x)/2;

end;

const

CF : Ftype = Sh;

Типізованим константам можна присвоювати інші значення в ході виконання програми, тому фактично вони являють собою зміні з початковими значеннями. Типізована константа набуває зазначене в її оголошенні значення, тобто ініціюється, лише один раз: до моменту початку роботи програми. При повторному вході в блок (процедуру або функцію), у якому вона оголошена, ініціація типізованої константи не відбувається і вона зберігає те значення, що мала до моменту виходу з блока.

Типізовані константи можуть бути будь-якого типу, крім файлів. Не можна також оголосити типізовану константу-запис, якщо хоча б одне з її полів є полем файлового типу.

Оскільки типізована константа фактично не відрізняється від змінної її не можна використовувати в якості значення при оголошенні інших констант або меж типу-діапазону.

1. Розділ типів. Типи даних в ТурбоПаскалі. Стандартні та користувацькі типи даних. Приклади.

Для обробки ЕОМ дані представляються у вигляді величин та їх сукупностей. З поняттям величини пов'язані така важлива характеристика, як її тип.

Тип визначає:

* можливі значення змінних, констант, функцій, виразів, що належать до даного типу;
* внутрішню форму представлення даних в ЕОМ;
* операції та функції, що можуть виконуватися відносно до величин, котрі належать до даного типу.

У мові Паскаль тип величини визначають заздалегідь. Всі змінні, що використовуються в програмі, повинні бути оголошені в розділі опису разом із зазначенням їх типу. Обов'язковий опис типу призводить до надмірності в тексті програми, але така надмірність є важливим допоміжним засобом розробки програм і розглядається як необхідна властивість сучасних алгоритмічних мов високого рівня.

Ієрархія типів у мові Паскаль наступна:

* Прості
  + Порядкові
    - Цілі
    - Логічні
    - Символьні
    - Перераховувані
    - Інтервальні
  + Дійсні
* Структуровані
  + Масиви
  + Рядки
  + Множини
  + Записи
  + Файли
* Покажчики

Прості типи даних

У таблиці наведені прості типи даних Турбо Паскаль, обсяг пам'яті, який необхідний для зберігання однієї змінної зазначеного типу, множина допустимих значень і допустимі операції.

|  |  |  |  |
| --- | --- | --- | --- |
| Ідентифікатор | Довжина (байт) | Діапазон значень | Операції |
| **Цілі типи** | | | |
| Integer | 2 | -32768..32767 | +, -, /, \*, Div, Mod, >=, <=, =, <>, <, > |
| Byte | 1 | 0..255 |
| Word | 2 | 0..65535 |
| Shortint | 1 | -128..127 |
| Longint | 4 | -2147483648..2147483647 |
| **Дійсні типи** | | | |
| Real | 6 | 2,9x10-39 - 1,7x1038 | +, -, /, \*, >=, <=, =, <>, <, > |
| Single | 4 | 1,5x10-45 - 3,4x1038 |
| Double | 8 | 5x10-324 - 1,7x10308 |
| **Логічний тип** | | | |
| Boolean | 1 | true, false | Not, And, Or, Xor, >=, <=, =, <>, <, > |
| **Символьний тип** | | | |
| Char | 1 | Всі символи коду ASCII | +, >=, <=, =, <>, <, > |

**Користувацькі типи ДАНИХ**

У Турбо Паскалі передбачений механізм створення нових типів, які прийнято називати користувача або конструюються. Їх можна створювати на основі [стандартних](http://ua-referat.com/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82) і раніше створених типів. [Опис](http://ua-referat.com/%D0%9E%D0%BF%D0%B8%D1%81) нових типів відбувається в розділі TYPE. Після цього можна в розділі Var створювати змінні цих типів. Також, можна відразу описувати новий тип при створенні змінної в розділі Var.

Користувацькі типи:

* перелічувальний (type Season =(Spring, Winter, Autumn, Summer);
* діапазонний(0..500, при описанні вказується мінімальне і максимальне значення типу через 2 точки)
* вказівниковий(крім стандартного типа Pointer)- значення вказівникових величин є адреси памяті

Type prtS = ^TS;

TS=record;

Inf: real;

link:prtS;

end;

* структурований

До структурованих типів відносять - тип-масив, тип-множина, тип-запис, файловий тип, об’єктний тип.

* процедурний

1. Операнди, вирази, операції. Класифікація операцій. Приорітет операцій. Приклади.

Вирази в програмуванні існує для визначення дій, які в матиматиці зазвичай описується формулами. Визази складаються з операцій і операндів. По кількості операндів операції діляться на унарні та бінарні. унарні - мають тільки один операнд, перед яким стоїть символ операції.

Операнди:

¨ Проста змінна

¨ Змінна з індексом

¨ Елемент запису

¨ Функція

¨ Множина

¨ Константа

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. 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;

• рядкові операції;

• операції над множинами

• операція @

1. 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 | цілочисельне 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. Винятком є лише операція ділення, результат якої завжди має

дійсний тип. Зміст арифметичних операцій аналогічний відповідним операціям

математики.

2. Операції відношення. Результатом операцій відношення є булевський тип. Вони можут застосовуватися до будь-яких типів

2.1. Арифметичне порівняння

1<2 à True

2 <> 2 à False

2.2. Логічне порівняння

False < True à True

2.3. Символьне порівняння (ASCII)

‘a’ < ‘z’ à True

2.4. Порівняння рядків (до першого неспівпадання)

‘Я ‘ > ‘Петя’

3. Логічні операції (булевські)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Not A | A and B | A or B |
| False | False | True | False | False |
| False | True | True | False | True |
| True | False | False | False | True |
| True | True | False | True | True |

4. Рядкові

Наприклад, конкатенація (+)

5. Операції над множинами

5.1. Перетин (\*)

5.2. Об’єднання (+)

5.3. Різниця (-)

5.4. =, <>

5.5. >= (є надмножиною)

5.6. <= (є підмножиною)

5.7. In (true – елемент належить множині, false – не належить)

6. Операція @

Операнди і операції формують вирази. Вираз – це сукупність операндів, поєднаних знаком операцій і круглими дужками. 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 (найвищий) | унарні операції |
| \*, /, div, mod, nd, shl, shr | другий | операції множення |
| +, -, or, xor | третій | операції додавання |
| =, <>, <, >,<=, >=, in | четвертий | операції відношення |

1) Операції з однаковим пріоритетом виконуються зліва направо.

2) Операнд, ща знаходиться між двома операціями з різними пріоритететами, зв’язаний с тією, що має вищий пріоритет.

3) Операнд, ща знаходиться між двома операціями з рівними пріоритетами, зв’язаний с тією, що ліворуч

4) Вираз в дужках обчислюється як окремий операнд.

1. Введення і виведення даних, форматування введення-виведення. Форми подання чисел в ТурбоПаскалі. Приклади.

**Введення даних**

Для введення вихідних даних найчастіше використовується процедури Read та ReadLn:

Read(<список елементів вводу>)

Readln (<список елементів вводу>)

<елементи вводу>::= <змінна> | <змінна з індексом> | <елемент запису>

ReadLn(A1,A2,...AK);

Процедура робить читання K значень вихідних даних і надає (присвоює) ці значення змінним А1, А2, ..., АК.

При введенні вихідних даних відбувається перетворення з зовнішньої форми представлення у внутрішню, яка, в свою чергу, визначається типом змінних. Змінні, що утворюють список введення, можуть належати або до цілого, або до дійсного, або до символьного типу. Читання вихідних даних логічного типу в мові Паскаль неприпустимо.

Значення вихідних даних можуть відокремлюватися один від одного пробілами і натисканням клавіш табуляції, а також клавіші Enter.

Числа, що вводяться, не можна розділяти комами.

**Виведення даних**

Для виведення результатів роботи програми на екран використовуються процедури:

Write(A1,A2,...AK);

WriteLn(A1,A2,...AK);

Перший з цих операторів здійснює вивід значень змінних А1, А2, ..., АК в один рядок на екрані. Другий оператор, на відміну від першого, не тільки здійснює вивід на екран певного рядку, але й переносить курсор вводу на наступний рядок екрану. Якщо процедура writeln використовується без параметрів, то вона просто здійснить перенесення курсору на наступний рядок.

Змінні, що складають список виводу, можуть відноситися до цілого, дійсного, символьного чи булевого типів. В якості елемента списку виведення крім імен змінних можуть використовуватися вирази й рядки.

Перелічувальні типи не виводяться!

Форма подання значень у полі виводу відповідає типу змінних та виразів: величини цілого типу виводяться як цілі десяткові числа, дійсного типу - як дійсні десяткові числа з десятковим порядком, символьного типу і рядки - у вигляді символів, логічного типу - у виді логічних констант TRUE і FALSE.

Оператор виводу дозволяє встановити ширину поля виводу для кожного елемента списку виводу. У цьому випадку елемент списку виводу має вид А:К, де А - вираз чи рядок, К - вираз або константа цілого типу. Якщо значення, що виводиться, займає в полі виводу менше позицій, ніж К, то перед цим значенням проставляються пробіли. Якщо виведене значення не вміщається в ширину поля К, то для цього значення додатково буде відведена необхідна кількість позицій.

Для величин дійсного типу елемент списку висновку може мати вигляд А:К:М, де А - змінна чи вираз дійсного типу, К - ширина поля виводу, М - число цифр дробової частини значення, яке виводиться. К і М – це вирази чи константи цілого типу. У цьому випадку дійсні значення виводяться у формі десяткового числа з фіксованою комою.

Приклад запису операторів виводу:

var rA, rB: Real;

iP,iQ:Integer;

bR, bS: Boolean;

chT, chV, chU, chW: Char;

begin

. . .

WriteLn(rA, rB:10:2);

WriteLn(iP, iQ:8);

WriteLn(bR, bS:8);

WriteLn(chT, chV, chU, chW);

end.

Перш за все, дані дійсного типу можуть подаватись у двох формах: з фіксованою точкою та плаваючою точкою (експоненційній формі). Перша форма подання чисел більш звична. В ній явно задана ціла та дробова частина, які відокремлені точкою Так, числа 2.729, -89.084109, 134 подані у формі з фіксованою точкою.

Експоненційна форма подає число у так званому нормалізованому вигляді: мантиси і порядку. Мантиса лежить у діапазоні [1 ; 10) і складається з 12 символів: однієї цифри на цілу частину, одного символу на десяткову точку і десяти цифр на дробову частину. Якщо дробова частина містить менше цифр, то решта заповнюється нулями. Порядок складається з чотирьох символів і починається літерою Е, після якої йде знак порядку та дві цифри – його значення. Приклад запису чисел поданий в таблиці

Запис дійсних чисел

|  |  |
| --- | --- |
| Форма з фіксованою точкою | Експоненційна форма |
| 1.4529 | 1.4529000000Е+00 |
| 39870 | 3.9870000000Е+04 |
| 0.000029 | 2.9000000000Е-05 |
|  |  |

1. Розділ операторів. Оператор присвоювання, оператор процедури, оператор переходу, складений оператор. Приклади.

Під операторами в мові Паскаль мають на увазі тільки опис дій. Оператори відокремлюються один від одного тільки крапкою з комою. Якщо оператор стоїть перед end, until або else, то в цьому випадку крапка з комою не ставиться.

**Розділ операторів** являє собою складений оператор, що міститься між службовими словами

Begin ……….. End.

<розділ операторів>::= begin <список операторів> end.

<список операторів> ::= <оператор> {; <оператор>}.

Прості оператори не містять в собі інших операторів:

1) оператор присвоєння

Загальний вигляд:

v : = a ;

тут v - змінна, a - вираз , : = - операція присвоювання . Вираз a може містити константи, змінні, назви функцій, знаки операцій та дужки.

Приклад. f : = 3 \* C + 2 \* sin ( x );

Присвоєння допускається для всіх типів змінних , за винятком типу файл.

В операції v : = a змінна v і вираз a повинні мати один і той же тип , а для інтервального типу - одну і ту ж підмножину значень.

2) оператор процедури

<оператор процедури>::= <ідентифікатор процедури> {(список фактичних параметрів)}

Ініціалізує виклик процедури.

3) оператор переходу

<оператор переходу>::= goto <мітка>

Оператор goto переходить при виконанні програми до певного оператора програми, перед яким знаходиться *мітка*. *Мітка* повинна бути описана в розділі опису міток тієї програми (процедури або функції), в якій вона використовується. Не можна перейти з однієї процедури або функції в іншу.

Необхідно, щоб в програмі існував оператор, позначений даною міткою. Вона записується перед оператором і відокремлюється від нього двокрапкою.

*Приклад*

label 1;

begin

...

goto 1;

...

1: WriteLn('Перехід до мітки 1');

end.

1. Умовний оператор. Вкладеність умовного оператора. Оператор вибору case. Особливості його використання. Порівняння роботи умовного оператора та оператора вибору. Приклади.

Під операторами в мові Паскаль мають на увазі тільки опис дій. Оператори відокремлюються один від одного тільки крапкою з комою. Якщо оператор стоїть перед end, until або else, то в цьому випадку крапка з комою не ставиться.

**Розділ операторів** являє собою складений оператор, що міститься між службовими словами

Begin ……….. End.

<розділ операторів>::= begin <список операторів> end.

<список операторів> ::= <оператор> {; <оператор>}.

Структурні оператори можуть містити інші оператори:

1) складений оператор

<складений оператор>::= begin <оператори> end;|.

Вживається тоді, коли синтаксис Pascal дозволяє один оператор, а необхідно виконати групу.

2) умовний оператор

<умовний оператор>::= if <логічний вираз> then <оператор 1> {else <оператор 2>}

1. Повна форма. В залежності від логічного виразу обирається одна з двох альтернативних гілок алгоритму. Якщо значення умови істинно (TRUE), то буде виконуватися *оператор 1*, записаний після ключового слова then. В іншому випадку буде виконуватись *оператор 2*, що слідує після слова else, при цьому *оператор 1* пропускається. Після виконання зазначених операторів програма переходить до виконанню команди, яка стоїть безпосередньо після оператора if.

Необхідно пам'ятати, що перед ключовим словом else крапка з комою ніколи не ставиться!

2. Неповна форма. else - частина в операторі if може бути відсутня:

Тоді, в разі невиконання логічної умови, керування відразу передається оператору, який стоїть у програмі після конструкції if.

Існує поняття вкладеності умовного оператора.

3) оператор вибору

Часто виникають ситуації, коли доводиться здійснювати вибір одного з декількох альтернативних шляхів виконання програми. Незважаючи на те, що такий вибір можна організувати за допомогою оператора if .. then, зручніше скористатися спеціальним оператором вибору. Його формат:

<оператор вибору>::= case <вираз-селектор> of <опис варіанту> {<опис варіанту>;} {else <оператор>} end;

<вираз-селектор>::= <вираз>

<опис варіанту>::=<мітка варіанту>:<оператор>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оператор умови

Умовні оператори призначені для вибору до виконання однієї з двох можливих дій (операторів) в залежності від деякої умови (при цьому одна з дій може бути відсутньою). Як умова використовується значення логічного виразу. Структура умовного оператора така:

**If <умова> then <оператор> else <оператор>;**

Даний оператор виконується так. Спочатку обчислюється значення умови, яка стоїть після службового слова **if**. Результат обчислення повинен мати булевий тип. Якщо значення виразу **true** (тобто умова виконується), то виконується оператор, який вказано після службового слова **then**. Якщо результат обчислень є **false**(тобто хиба), виконується оператор, що стоїть після слова **else**. Можлива інша скорочена форма умовного оператора:

**If <умова> then <оператор>;**

У цьому випадку частина умовного оператора, що йде за словом **else** відсутня. Тоді при хибній умові керування негайно передається наступному за умовним оператору.

Наведемо декілька прикладів.

Програма обчислення максимального з двох чисел.

**Program Maximum;**

**var**

**x, y, m : integer;**

**begin**

**write(‘введіть числа x та y ’);**

**readln(x,y);**

**if x<y then m:=y else m:=x;**

**writeln(‘з двох чисел більше ’, m);**

**end.**

Програма обчислення модуля числа.

**Program Modul;**

**var**

**x : real;**

**begin**

**write(‘введіть число x ’);**

**readln(x);**

**if x<0 then x:=-x;**

**writeln(‘модуль числа ’, x, ‘дорівнює’, x);**

**end.**

При використанні вкладених умовних операторів, службове слово **else** завжди асоціюється з найближчим до нього по тексту словом **if**, яке ще не зв'язане із словом**else**.

***Зауваження 1.*** *В <умова> можна використовувати також складні умови, які розділяються логічними операціями* ***and****,* ***or****,* ***xor*** *чи* ***not****. При цьому кожна умова повинна бути в дужках. Наприклад:*

**If (i>1) or (i<10) then y:=5 else y:=y-2.**

***Зауваження 2.*** *В <оператор> можна використовувати не один оператор, а декілька, але їх треба об'єднати в один складений оператор (взяти в операторні дужки begin-end).*

Наприклад:

**If i>1 then**

**Begin**

**y:=y-5;**

**x:=y+8;**

**end;**

1. Оператори повтору. Порівняння роботи операторів повтору. Процедури, що змінюють хід роботи циклічного оператора. Приклади.

У більшості завдань, що зустрічаються на практиці, необхідно здійснювати багаторазове виконання однієї дії. Ділянка обчислювального процесу, що багаторазово повторюється, називається **циклом**.

Якщо заздалегідь відомо кількість необхідних повторень, то цикл називається **арифметичним**. Якщо ж кількість повторень заздалегідь невідомо, то говорять про **ітераційний** цикл.

В ітераційних циклах відбувається перевірка деякої умови і в залежності від результату цієї перевірки відбувається або вихід з циклу, або повторення виконання тіла циклу. Якщо перевірка умови відбувається перед виконанням блоку операторів, то такий ітераційний цикл називається циклом з **передумовою** (цикл "поки"), а якщо перевірка проводиться після виконання тіла циклу, то це цикл з **післяумовою** (цикл "до").

Особливість цих циклів полягає в тому, що тіло циклу з післяумовою завжди виконується хоча б один раз, а тіло циклу з передумовою може жодного разу не виконатися. В залежності від умови задачі необхідно використовувати той чи інший вид ітераційних циклів.

Арифметичні цикли

Синтаксис:

for змінна := значення 1 to значення 2 do оператор

або

for змінна := значення 1 downto значення 2 do оператор

Оператор for викликає *оператор*, що знаходиться після слова do, по одному разу для кожного значення в діапазоні від *значення 1* до *значення 2*.

Змінна циклу, початкове і кінцеве значення повинні мати порядковий тип. Зі словом to, значення змінної циклу збільшується на 1 при кожній ітерації циклу. Зі словом downto, значення змінної циклу зменшується на 1 при кожній ітерації циклу. Не слід самостійно змінювати значення керуючої змінної всередині циклу.

Як і у випадку використання оператора умовного переходу, слід пам'ятати, що синтаксис мови допускає запис тільки одного оператора після ключового слова do, тому, якщо ви хочете в циклі виконати групу операторів, обов'язково треба об'єднати їх у складений оператор (обмежувати операторними дужками begin .. . end). В іншому випадку буде зроблена логічна помилка програми.

*Приклад 1. Квадрати чисел від 2-х до 10-и.*

for x:=2 to 10 do WriteLn(x\*x);

*Приклад 2. Латинський алфавіт.*

for ch:='A' to 'Z' do Writeln(ch);

*Приклад 3. Використання циклу з downto.*

for i:=10 downto 1 do WriteLn(i);

*Приклад 4. Використання складеного оператора.*

for x:=1 to 10 do begin

y:=2\*x+3;

WriteLn('f(',x,')=',y);

end;

Ітераційні цикли з передумовою

Синтаксис:

while вираз do оператор

Оператор після do буде виконуватися доки логічний вираз приймає істинне значення (True). Логічний вираз є умовою поновлення циклу. Його істинність перевіряється щораз перед черговим повторенням оператора циклу, який буде виконуватися лише до тих пір, поки логічне вираз істинний. Як тільки логічний вираз приймає значення неправда (False), здійснюється перехід до оператора, наступного за while.

Істинність виразу перевіряється до виконання оператора, тому якщо він із самого початку буде мати значення неправда (False), то оператор не виконається жодного разу.

Тут також варто пам'ятати, що дозволяється використовувати тільки один оператор після ключового слова do. Якщо необхідно виконати групу операторів, то варто використовувати складений оператор.

*Приклад.*

eps:=0.001;

while x > eps do x:=x/2;

Ітераційні цикли з післяумовою

Синтаксис:

repeat

оператор;

оператор;

...

оператор

until вираз

Оператори між словами repeat і until повторюються, поки логічний вираз є не істинним (False). Як тільки логічний вираз набуває значення істина (True), відбувається вихід із циклу.

Так як вираз оцінюється після виконання операторів, то в будь-якому випадку оператори виконаються хоча б один раз.

*Приклад.*

repeat

WriteLn('Введіть додатнє число');

ReadLn(x);

until x>0;

Оператори завершення циклу

Для всіх операторів циклу вихід із циклу здійснюється як внаслідок природного закінчення оператора циклу, так і за допомогою операторів переходу і виходу.

У версії Турбо Паскаль 7.0 визначені стандартні процедури:

Break

Continue

Процедура Break виконує безумовний вихід із циклу. Процедура Continue здійснює перехід до початку чергової ітерації циклу.

Зауважимо, що хоча й існує можливість виходу з циклу за допомогою оператора безумовного переходу goto, реалізовувати її не бажано. У всіх випадках можна скористатися спеціально призначеними для цього процедурами Break і Continue.

1. Масиви. Основні визначення. Описання масивів. Операції над масивами. Приклади.

**Масиви** – це сукупності однотипних елементів. Вони мають наступні характеристики:

* кожен компонент масиву може бути явно позначений і до нього є прямий доступ;
* число компонентів масиву визначається при його описі і надалі не міняється.

Для позначення компоненту масиву використовується ім’я змінної-масиву і так звані індекси, які вказують на певний елемент. Тип індексу може бути тільки порядковим (крім longint). Найчастіше використовується інтервальний тип (діапазон).

Опис типу масиву задається наступним чином:

type

ім’я типу = array[ список індексів ] of тип

Тут ім’я *типу* – правильний ідентифікатор; ,*список індексів* – список одного або декількох індексних типів, розділених комами; *тип* – будь-який тип даних.

Вводити і виводити масиви можна тільки по одному елементу за раз.

*Приклад 1. Введення і виведення одновимірного масиву.*

Const

n = 5;

type

mas = array[1..n] of integer;

var

a: mas;

i: byte;

begin

writeln(‘введіть елементи масиву’);

for i:=1 to n do readln(a[i]);

writeln(‘вивід елементів масиву:’);

for i:=1 to n do write(a[i]:5);

end.

Визначити змінну як масив можна і безпосередньо при її описі, без попереднього опису типу масиву, наприклад:

var a,b,c: array[1..10] of integer;

Якщо масиви a і b описані як:

var

a = array[1 .. 5] of integer;

b = array[1 .. 5] of integer;

то змінні a і b будуть розрізнятись як змінні різних типів. Для забезпечення сумісності описуйте змінні через попередній опис типу.

Якщо типи масивів ідентичні, то в програмі один масив може бути привласнений іншим. У цьому випадку значення всіх змінних одного масиву буде надано відповідним елементам другого масиву.

Разом з тим, над масивами не визначені операції відношення. Порівнювати можна лише окремі елементи масивів.

Так як тип, що йде за ключовим словом of в описі масиву, - будь-який тип Турбо Паскаль, то він може бути і іншим масивом. Наприклад:

type

mas = array[1 .. 5] of array[1 .. 10] of integer;

Такий запис можна замінити більш компактним:

type

mas = array [1 .. 5, 1 .. 10] of integer;

1. Рядки. Різні способи подання рядків. Поняття довжини рядка. Операції над рядками. Приклади.

Рядковий тип даних

Для обробки інформації, поданої в рядковому вигляді, в Турбо Паскаль введений рядковий тип даних. Рядком у Паскалі називається певна послідовність символів. Кількість символів послідовності називається довжиною рядка. Синтаксис:

var s: string[n];,

var s: string;

n - максимально можлива довжина рядка - ціле число в діапазоні від 1 до 255. Якщо цей параметр опущений, то за умовчанням він приймається рівним 255.

Змінні, що описані як рядкові і мають різну довжину можна привласнювати одна одній. Варто зазначити, що при спробі привласнити більш короткій змінній довшу зайві символи будуть відкинуті.

Вирази типу char можна привласнювати будь-яким рядковим змінним.

У Турбо Паскаль є простий доступ до окремих символів рядковий змінної: запис st[i] варто розуміти як елемент масиву st з індексом i. Наприклад, якщо st - це 'Рядок', то st [1] - це 'Р', st [2] - це 'я', st [3] - 'д' і так далі.

Над рядковими даними визначена операція злиття (конкатенації), що позначається знаком +. Наприклад:

a: = 'Turbo';

b: = 'Pascal';

c: = a + b;

У цьому прикладі змінна c набуде значення 'TurboPascal'.

Крім злиття над рядками визначені операції порівняння <,>, =, <>, <=, >=. Два рядки порівнюються по символу за раз, зліва направо, по кодам символів. Якщо один рядок менше іншого по довжині, то бракуючі символи коротшого рядка будуть замінені символом з кодом 0.

1. Записи. Фіксовані записи, їх описання в програмі. Операції над записами. Оператор приєднання with. Переваги його використання. Приклади.

Запис являє собою сукупність обмеженого числа логічно зв'язаних компонент, що належать до різних типів. Компоненти запису називаються полями, кожне з яких визначається ім'ям. Поле запису містить ім'я поля, після якого, через двокрапку, вказується тип цього поля. Поля запису можуть відноситися до будь-якого типу, допустимого в мові Паскаль, за винятком файлового.

Опис запису в мові Паскаль здійснюється за допомогою службового слова record, за яким слідує опис компоненти запису. Завершується опис запису службовим словом end.

Наприклад, телефонний довідник містить прізвища і номери телефонів, тому окремий рядок в такому довіднику зручно представити у вигляді наступного запису:

type TRec = Record

FIO: String[20];

TEL: String[7]

end;

var rec: TRec;

Звернення до запису в цілому допускається тільки в операторах присвоєння, де ліворуч і праворуч від знака присвоювання використовуються імена записів однакового типу. У всіх інших випадках оперують окремими полями записів. Щоб звернутися до окремої компоненти запису, необхідно задати ім'я запису і через крапку вказати ім'я потрібного поля, наприклад:

rec.FIO, rec.TEL

Таке ім'я називається **складеним**. Компонентою запису може бути іще один запис, у такому випадку складене ім'я буде містити не два, а більшу кількість імен.

Звернення до компонентів записів можна спростити, якщо скористатися оператором приєднання with.

Він дозволяє замінити складені імена, що характеризують кожне поле на імена полів, а ім'я запису визначити в операторі приєднання:

with rec do оператор;

Тут rec - ім'я запису, *оператор* - оператор, простий або складений. Оператор являє собою область дії оператора приєднання, у межах якої можна не використовувати складені імена. Наприклад для нашого випадку:

with rec do begin

FIO:='Козак С.В.';

TEL:='2324231';

end;

Така алгоритмічна конструкція повністю ідентична наступній:

rec.FIO:='Козак С.В.';

rec.TEL:='2324231';

Щоб зробити звернення до полів запису більш коротким використовується оператор приєднання with :

with < змінна> do

begin

< оператори >

end ;

< змінна> - ім'я змінної типу запис ,

< оператори > - будь-які оператори Паскаля .

Усередині блоку операторів можна при зверненні до полів запису опускати ім'я змінної . Замість невказаного імені за замовчуванням буде додано ім'я з першої строчки оператора. наприклад:

with s do

begin

x : = 64 ; { Скорочена звернення до поля x записи s , те ж що sx }

y : = 32 ; { Скорочена звернення до полю sy }

Name: = ' Saratov ';

zy : = y +20 ; { Можна звертатися до полів інших записів , але доведеться

використовувати повну форму }

{ А ось так робити не можна }

xy : = y +20 ; { Звертатися до полів запису з ім'ям <em> x </ em > всередині

такого оператора не можна через збіги імен записи

x і одного з полів запису s }

end ;

1. Записи. Варіантні записи, їх описання в програмі. Приклади.

Особливим різновидом записів є записи з варіантами, які оголошуються з використанням зарезервованого слова case. За допомогою записів з варіантами ви можете одночасно зберігати різні структури даних, які мають загальну спільну частину, що однакова у всіх структурах, але відрізняються деякими невеличкими частинами

Наприклад, побудуємо запис, в якому ми будемо зберігати дані про деяку геометричну фігуру (відрізок, трикутник, коло).

type

TFigure = record

type\_of\_figure: string[10];

color\_of\_figure: byte;

...

case integer of

1: (x1,y1,x2,y2: integer);

2: (a1,a2,b1,b2,c1,c2: integer);

3: (x,y: integer; radius: word);

end;

var figure: TFigure;

Таким чином, у змінній figure ми можемо зберігати дані як про відрізок, так і про трикутник або коло. Треба лише в залежності від типу фігури звертатися до відповідних полів запису.

Зауважимо, що індивідуальні поля для кожного з типів фігур займають один адресний простір пам'яті, а це означає, що одночасне їх використання неможливе.

У будь-якому записі може бути тільки одна варіантна частина. Після закінчення варіантної частини в записі не можуть з'являтися ніякі інші поля. Імена полів повинні бути унікальними в межах того запису, де вони оголошені.

1. Процедури та функції. Описання процедур та функцій. Область дії ідентифікаторів. Глобальні та локальні імена. Приклади.

Процедури та функції

У мові Паскаль, як і в більшості мов програмування, передбачені засоби, що дозволяють оформлювати допоміжний алгоритм як підпрограму.

У мові Паскаль є два види підпрограм – **процедури** та **функції**.

Процедури і функції розміщують в розділі описів програми. Для обміну інформацією між процедурами і функціями і іншими блоками програми існує механізм вхідних і вихідних параметрів. **Вхідними** параметрами називають величини, які передаються із блоку викликів до підпрограми (вихідні дані для підпрограми), а **вихідними** – ті, що передаються з підпрограми до блоку викликів (результат роботи підпрограми).

Одна і та ж підпрограма може викликатися неодноразово, виконуючи одні і ті ж дії з різними наборами вхідних даних. Параметри, що використовуються під час запису тексту підпрограми в розділ описів, називають **формальними**, а ті, що використовуються під час її виклику –**фактичними**.

Опис і виклик процедур і функцій

Структура опису процедур і функцій деякою мірою схожа на структуру Паскаль-програми: у них також є заголовок, розділ описів і виконувана частина. Розділ описів містить ті ж підрозділи, що і розділ описів програми: описи констант, типів, міток, процедур, функцій, змінних. Виконувана частина містить, власне, оператори процедур.

Формат опису процедури має вигляд:

procedure ім’я процедури (формальні параметри);

розділ опису процедур

begin

виконувана частина процедури

end;

Формат опису функції:

function ім’я функції (формальні параметри): тип результату;

розділ опису функції

begin

виконувана частина функції

end;

Формальні параметри в заголовках процедур і функцій записують у вигляді:

var ім’я параметру: ім’я типу

і відокремлюються один від одного крапкою з комою. Ключове слово var може бути відсутнім (про це далі). Якщо параметри однотипні, то їх імена можна перераховувати через кому, вказуючи загальне для них ім’я типу. Під час опису параметрів можна використовувати лише стандартні імена типів, або імена типів, визначені за допомогою команди type. Список формальних параметрів може бути відсутнім.

Виклик процедури здійснюється оператором, що має наступний формат:

ім’я процедури (список фактичних параметрів);

Список фактичних параметрів – це їх перерахування через кому. Під час виклику фактичні параметри ніби підставляються замість формальних, що стоять на тих же місцях у заголовку. Таким чином відбувається передача вхідних параметрів, потім виконуються оператори виконуваної частини процедури, після чого відбувається повернення в блок викликів. Передача вихідних параметрів відбувається безпосередньо під час роботи виконуваної частини.

Виклик функції в Турбо Паскаль може відбуватися в аналогічний спосіб, окрім того є можливість здійснити виклик всередині якогось виразу. Зокрема ім’я функції може стояти в правій частині оператора присвоєння, в розділі умов оператора if і т.д.

До повернення в блок викликів необхідно додати наступну команду в виконувану частину функції, для того, щоб передати в блок викликів вихідне значення функції

ім’я функції: = результат;

Під час виклику процедур і функцій необхідно дотримуватися наступних правил.

* кількість фактичних параметрів має збігатися з кількістю формальних;
* відповідні фактичні та формальні параметри повинні збігатися по порядку

проходження і по типу.

Зауважимо, що імена формальних і фактичних параметрів можуть збігатися. Це не призводить до проблем, так як відповідні їм змінні все одно будуть різні із-за того, що зберігаються в різних областях пам’яті. Крім того, всі формальні параметри є тимчасовими змінними – вони створюються в момент виклику підпрограми і знищуються в момент виходу з неї.

Розглянемо використання процедури на прикладі програми пошуку максимуму з двох цілих чисел.

Var x,y,m,n: integer;

procedure MaxNumber(a,b: integer; var max: integer);

begin

if a>b then max:=a else max:=b;

end;

begin

write(‘Введіть x,y ‘);

readln(x,y);

MaxNumber(x,y,m);

MaxNumber(2,x+y,n);

writeln(‘m=’,m,’n=’,n);

end.

Аналогічну задачу, але уже з використанням функцій, можна вирішити так:

var x,y,m,n: integer;

function MaxNumber(a,b: integer): integer;

var max: integer;

begin

if a>b then max:=a else max:=b;

MaxNumber := max;

end;

begin

write(‘Введіть x,y ‘);

readln(x,y);

m := MaxNumber(x,y);

n := MaxNumber(2,x+y);

writeln(‘m=’,m,’n=’,n);

end.

Локальні і глобальні ідентифікатори

Використання процедур та функцій в Паскалі тісно пов’язано з деякими особливостями роботи з ідентифікаторами (іменами) в програмі. Зокрема, не всі імена завжди доступні для використання. Доступ до ідентифікатора в конкретний момент часу визначається тим, у якому блоці він описаний.

Імена, описані в заголовку або розділі описів процедури або функції називають **локальними**для цього блоку. Імена, описані в блоці, що відповідає всій програмі, називають глобальними. Слід пам’ятати, що формальні параметри процедур і функцій завжди є локальними змінними для відповідних блоків.

Основні правила роботи з глобальними і локальними іменами можна сформулювати так:

* Локальні імена доступні (вважаються відомими, «видимими») тільки всередині того блоку, де вони описані. Сам цей блок та всі інші, що вкладені в нього, називають **областю видимості** для цих локальних імен.
* Імена, описані в одному блоці, можуть збігатися з іменами з інших: тих, що утримують даний блок і тих, що вкладені у нього. Це пояснюється тим, що змінні, описані в різних блоках (навіть якщо вони мають однакові імена), зберігаються в різних областях оперативної пам’яті.

Глобальні імена зберігаються в області пам’яті, що називається **сегментом даних** (статичним сегментом) програми. Вони створюються на етапі компіляції і можуть бути використані будь-коли, за увесь час роботи програми.

На відміну від них, локальні змінні зберігаються в спеціальній області пам’яті, яка називається**стек**. Вони є тимчасовими, оскільки створюються в момент входу в підпрограму і знищуються при виході з неї.

Ім’я, що описане в блоці, «закриває» співпадаючі з ним імена з блоків, що містять даний. Це означає, що якщо в двох блоках, один з яких міститься всередині іншого, є змінні з однаковими іменами, то після входу у вкладений блок робота відбуватися з локальною для даного блоку змінною. Змінна з тим же ім’ям, описана в зовнішньому блоці, стає тимчасово недоступною і це продовжується до моменту виходу з вкладеного блоку.

Рекомендується всі імена, що мають в підпрограмах суто внутрішнє, допоміжне призначення, робити локальними. Це запобігає змінам глобальні об’єкти з такими ж іменами.

Формат опису: Procedure <ім'я> [(формальні параметри)]; Приклад:

Procedure Korrekt;

Procedure Sort (A:byte);

Ім'я процедури - ідентифікатор, унікальний у межах програми. Тіло процедури являє собою локальний блок, за структурою аналогічний програмі:

Procedure <ім'я> [(формальні параметри)];

[<розділи описів>;]

begin

<розділи операторів>

end;

Зверніть увагу, що як формальні параметри, так і розділ описів у процедурі може бути відсутній. Щоб звернутися до процедури, треба використати оператор виклику процедури. Він складається з ідентифікатора (імені) процедури і списку фактичних параметрів, що відділені один від одного комами і знаходяться у круглих дужках. Якщо процедурі не передається ніяких параметрів, то фактичні параметри не вказуються. Формат виклику процедури: <ідентифікатор> [(фактичні параметри)]; Параметри забезпечують механізм заміни, який дозволяє виконувати процедуру з різними початковими даними. Між фактичними параметрами в операторі виклику процедури і формальними параметрами у заголовку опису процедури встановлюється взаємо-однозначна відповідність у результаті їхнього перебору зліва направо. Фактичні параметри за кількістю і типами повинні дорівнювати кількості і типам формальних параметрів. Параметри, за допомогою яких здійснюється обмін значеннями змінних між підпрограмами та програмою, що їх викликає, можуть мати будь-який тип, в тому числі структурований.

1. Процедури та функції. Передача параметрів: параметри-змінні, параметри-значення, параметри-константи, нетипізовані параметри, правила їх передачі.

Передача параметрів

У стандарті мови Паскаль передача параметрів може здійснюватися в два способи – по значенню і по посиланню. Параметри, що передаються по значенню, називають **параметрами-значеннями**, що передаються по посиланню – **параметрами-змінними**. Останні відрізняються тим, що в заголовку процедури (функції) перед ними ставиться службове слово var.

Якщо використовується перший спосіб (передача за значенням) значення фактичних параметрів копіюється у відповідні формальні параметри. При зміні цих значень в ході виконання процедури (функції) вихідні дані (фактичні параметри) змінитися не можуть. Тому таким способом передають дані тільки із блоку викликів в підпрограму (тобто вхідні параметри). При цьому в якості фактичних параметрів можна використовувати і константи, і змінні, і вирази.

Якщо використовується другий спосіб (передача по посиланню) всі зміни, що відбуваються в тілі процедури (функції) з формальними параметрами, призводять до негайних аналогічним змінам відповідних їм фактичних параметрів. Зміни відбуваються із змінними блоку викликів, тому по посиланню передаються вихідні параметри. В разі виклику відповідні їм фактичні параметри можуть бути тільки змінними.

Вибір способу передачі параметрів при створенні процедури (функції) відбувається у відповідності до сказаного вище: вхідні параметри потрібно передавати за значенням, а вихідні – по посиланню. Практично це зводиться до розстановки в заголовку процедури (функції) зарезервованого слова var перед всіма параметрами, які позначають результат роботи підпрограми. Однак, у зв’язку з тим, що функція повертає тільки один результат, в її заголовку використовувати параметри-змінні не рекомендується.

Існують два типи параметрів: параметр-значення; параметр-змінна. Група параметрів, перед якими відсутнє зарезервоване слово Var, називається параметрами-значеннями. Наприклад, в описі Procedure Korrect(S,K:real) S і K - параметри-значення. Формальний параметр-значення обробляється, як локальна стосовно процедури або функції, змінна. Зміни формальних параметрів-значень не впливають на відповідні значення фактичних параметрів. Група параметрів, перед якими знаходиться ключове слово Var, називається параметрами-змінними. Наприклад, в описі Procedure Obr(Var A,B:integer); А та В - параметри-змінні. Параметр-змінна використовується в тому випадку, якщо значення повинно бути передане з процедури в блок, що її викликає. При активізації процедури або функції формальний параметр-змінна заміщується фактичною змінною, а тому будь-які зміни в значенні формального параметра-змінної відбиваються на фактичному параметрі. І в тому, і в іншому випадку тип фактичного параметра повинен збігатися з типом формального. Якщо формальний параметр має рядковий тип, йому надається атрибут довжини, рівний 255, а тому і фактичний параметр в цьому випадку повинен також мати рядковий тип з атрибутом довжини, що дорівнює 255. У якості параметра-змінної може використовуватися будь-який тип, в тому числі файловий.

Відкриті масиви та рядки

За правилами Borland Pascal розмірність будь-якого масиву, переданого як параметр, повинна бути визначена. Отже, без використання спеціальних засобів застосування процедур і функцій, що мають параметри-масиви, істотно обмежується. Щоб зняти обмеження розмірності для параметрів - одночасних масивів, можна використовувати відкриті масиви.

Відкритий масив - це конструкція опису типу масиву без вказівки типу індексів.

Таке визначення можливо тільки при описі формальних параметрів підпрограми. Застосовуючи відкриті масиви, слід пам'ятати, що [індекси](http://ua-referat.com/%D0%86%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B8) параметрів, описаних як відкриті масиви, завжди починаються з нуля. Реальна кількість елементів фактичного параметра масиву можна визначити двома способами. По-перше - передати через додаткові параметри, по-друге - використовувати спеціальні функції.

High (<ідентифікатор масиву>) - для звичайного масиву повертає верхню межу індексу масиву, для відкритого - максимальне значення індексу.

Low (<ідентифікатор масиву>) - для звичайного масиву повертає нижню межу індексу масиву, для відкритого - нуль.

1.1.4 нетипізований параметри

У Borland Pascal допускається використовувати параметри, тип яких не вказаний. Такі параметри можуть передаватися в підпрограму тільки за посиланням, так як в цьому випадку в підпрограму реально передається адреса параметра.

Безумовно, для того щоб підпрограма могла виконувати будь-які дії з цим параметром, вона повинна якось призначити йому тип.

Для приведення нетипізовані параметри до певного типу можна використовувати:

Ø Автовизначення [перетворення](http://ua-referat.com/%D0%9F%D0%B5%D1%80%D0%B5%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F) типів;

Ø Накладене опис змінної певного типу.

При автовизначенням перетворення типів тип вираження вказують явно, наприклад:

Procedure Proc (Var: а);

b: = Integer (a) + 10;

Для накладення змінної певного типу використовують опис з absolute, наприклад:

Procedure Proc (Var: а);

Var r: real absolute a;

При цьому змінна r виявляється в пам'яті розміщеної у тому ж місці, що і нетипізовані параметри а, і, [відповідно](http://ua-referat.com/%D0%92%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D1%8C), будь-яка зміна r призведе до зміни а.

1. Множини. Описання множин. Операції над множинами, їх приорітет. Приклади.

**Множина** - структурований тип даних простих типів, який можна розглядати, як єдине ціле.

Елементи множини повинні належати одному типу(однобайтовому), який називається базовим. Максимальна кількість значень базового типу множини називається його потужністю. Кількість елементів множини від 0-255.

Описання:

|  |  |  |
| --- | --- | --- |
| Типізовані константи:  Const M: set of char =[‘a’,’b’,’c’]; | Введення з клавіатури:  Var A: set of char;  K:char;  Begin  For i:=1 to 5 do Begin  Read(k); A:=A+[k];  End; End. | Присвоєння:  Var S: set of byte;  Begin  S:=[1..3];  End. |

Операції над множинами в порядку їх пріоритету:

1.Перетин \*

2.Об’єднання +

3.Різниця -

4.Порівняння =,<>,<=,=>

5.Приналежності in

Задача підрахувати кількість цифр у текстовому файлі.

Program z1;

Type Charset= set of char;

Const digits : charset = [‘0’..’9’];

Var F: Text;

Ch: char;

K: word;

Begin

Assign(F, ‘f.dat’);

Reset(F);

While not eof F do

Begin

Read(ch);

If ch in digits then k:=k+1;

end;

Writeln(‘Кількість цифр’,k);

End.

Виведення множини:

Var M: set of char;

Begin

M:=[ ];

For i:=’a’ to ‘z’ do

If i in M then write(i);

1. Рекурсія. Випереджаюче описання. Приклади.

**Рекурсія** - це такий спосіб організації обчислювального процесу, при якому підпрограма в ході виконання складових її операторів звертається сама до себе.

Розглянемо класичний приклад - обчислення факторіала. Програма отримує від компонента edinput ціле число n і виводить в компонент lboutput значення N!, яке обчислюється за допомогою рекурсивної функції Factorial.

При виконанні правильно організованої рекурсивної підпрограми здійснюється багаторазовий перехід від деякого поточного рівня організації алгоритму до нижнього рівня послідовно доти, поки, нарешті, не буде отримано тривіальне рішення поставленого завдання. У нашому випадку рішення при n = 0 тривіально і використовується для зупинки рекурсії.

program Factorial;

var

n : Integer;

function Fact (i : integer) : Longint;

begin

if i = 1 then Fact := 1

else Fact := i\* Fact(i-1);

end;

begin

write (‘Введіть число n: ’);

Readln (n);

writeln (‘ n! = ’, Fact(n));

end.

Рекурсивна форма організації алгоритму зазвичай виглядає витонченіше ітераційної і дає більш компактний текст програми, але при виконанні, як правило, повільніше і може викликати переповнення стека (при кожному вході в підпрограму її локальні змінні розміщуються в організованій особливим чином області пам'яті, званої програмним стеком).

Рекурсивний виклик може бути непрямим. У цьому випадку підпрограма звертається до себе опосередковано, шляхом виклику іншої підпрограми, в якій міститься звернення до першої, наприклад:

procedure A(i: Byte);

begin

В(i);

end;

procedure В(j: Byte);

begin

A(j);

end;

Якщо строго слідувати правилу, згідно з яким кожен ідентифікатор перед вживанням повинен бути описаний, то таку програмну конструкцію використовувати не можна. Щоб такого роду виклики стали можливі, вводиться **випереджаючий опис**:

procedure В(j: Byte); forward;

procedure A(i: Byte);

begin

В(i);

end;

procedure B;

begin

A(j);

end;

Як бачимо, випереджаюче опис полягає в тому, що оголошується лише заголовок процедури В, а її тіло замінюється стандартною директивою Forward. Тепер у процедурі а можна використовувати звернення до процедури В - адже вона вже описана, точніше, відомі її формальні параметри, і компілятор може правильним чином організувати її виклик. Зверніть увагу: тіло процедури в починається заголовком, в якому вже не вказуються описані раніше формальні параметри.

1. Файли. Класифікація файлів. Операції над файлами. Стандартні процедури та функції для роботи з файлами. Відмінности файлів від інших структур даних.

Файл являє собою іменовану послідовність однотипних елементів, розміщених на зовнішньому пристрої. Для виконання операцій з конкретним файлом в програмі зазвичай використовується так звана файлова змінна або логічний файл. Залежно від типу елементів розрізняють текстові, типізовані і нетипізовані файли.

Текстовий файл можна розглядати як послідовність символів, розбиту на рядки довжиною від 0 до 256 символів. Текстовий файл - це файл, компонентами якого є символьні рядки змінної довжини, що закінчуються спеціальним маркером кінця рядка Для опису використовується стандартний тип Text:

var

F: text; {F - файлова змінна}

До типізований файлів відносяться файли строго певного типу(прямого, послідовного доступу). Найчастіше це файли, що складаються з записів. Вони застосовуються для створення різних баз даних.Типізований файл - це файл, всі компоненти якого одного типу, заданого при оголошенні файлової змінної. Компоненти файлу зберігаються на диску у внутрішньому (двійковому) форматі і нумеруються з 0. Якщо подивитися такий файл будь-яким текстовим редактором, то можна розпізнати тільки символьну інформацію, на місці ж чисел у файлі будуть розташовуватися прогалини або символи псевдографіки. Стандартне завдання в програмі такий файлової змінної здійснюється наступним чином:

var F: file of integer;

Нетипізований називають файли, оголошені без вказівки типу його компонентів. Операції читання і запису з такими файлами здійснюються блоками. Відсутність типу компонента робить ці файли сумісними з будь-якими іншими, а виконання вводу / виводу блоками дозволяє організувати високошвидкісний обмін даними між диском і пам'яттю. Нетипізовані файли, як і типізовані, допускають організацію прямого доступу. Для визначення в програмі нетипізованого файлу служить зарезервоване слово File:

var UntypedFile: File;

Процедури/функції:

1. Assign- зв'язує змінну файлого типу з фізичним файлом.
2. Read- зчитує елемент файлу
3. Eof- ознака кінця файлу
4. Write- записує елемент у файл
5. Reset- відкриває файл у режимі читання
6. Rewrite- відкриває файл у режимі редагування
7. Close- закриває файл
8. IOResult - перевірка існування файлу.
9. Rename-переменування
10. Erase- видалення файлу

Відмінність від масиву:

1. Файл має постійну пам'ять, масив використовує оперативну.
2. Масив має фіксований розмір, розмір файлу змінний.
3. Безліч кількість елементів, в масиві 64К
4. Нумерація елементів файлу з 0 до ознаки кінця файлу, в масиві по індексам елементів
5. Типізовані файли прямого доступу. Типізовані файлі послідовного доступу. Приклади.

Файл послідовного доступу це файли доступ до елементів яких відбувається почергово зчитування з 0го елемента. Використовується стандартні процедури.

Процедури/функції:

Assign- зв'язує змінну файлого типу з фізичним файлом.

Read- зчитує елемент файлу

Eof- ознака кінця файлу

Write- записує елемент у файл

Reset- відкриває файл у режимі читання

Rewrite- відкриває файл у режимі редагування

Close- закриває файл

IOResult - перевірка існування файлу.

Файл прямого доступу - доступ можна здійснювати до будь-якого елементу файлу, в залежності від вибраної позиції за допомогу спеціальної процедури.

Процедури:

1. Тіж самі, що і для послідовних файлів та
2. FilePos - повертає номер поточної позиції, при рахунку від 0
3. FileSize- повертає поточний ромір файлу (кількість елементів при рахунку від 1);
4. Seek - переміщує вказівник на елемент з заданим номером, при рахунку від 0
5. Truncate - обрізає розмір файлу до поточної позиції
6. Текстові файли. Нетипізовані файли. Приклади.

Текстовий файл можна розглядати як послідовність символів, розбиту на рядки довжиною від 0 до 256 символів. Текстовий файл - це файл, компонентами якого є символьні рядки змінної довжини, що закінчуються спеціальним маркером кінця рядка Для опису використовується стандартний тип Text:

Var F: text; {F - файлова змінна}

Робота з текстовими файлами.

Для роботи з текстовими файлами використовують спеціальні процедури

і функції.

1. Функція EOLn - визначає кінець рядка

2. Процедура Read ([Var f: text;] v1, v2, ... vn) - забезпечує введення символів, рядків і чисел. Якщо перед виконанням читання покажчик файлу досяг кінця черговий рядка, то результатом читання буде символ # 13, а якщо був досягнутий кінець файлу, то - символ # 26.

3. Процедура ReadLn ([Var f;] v1, v2,, .., vn) - також забезпечує введення символів, рядків і чисел. Процедура використовує ті ж правила введення, що і процедура Read, але після читання останньої змінної решта рядка до маркера кінця рядка пропускається, так що наступне звернення до ReadLn або Read почнеться з першого символу нового рядка.

4. Процедура Write ([Var f;] v1, v2, ..., vn) - забезпечує виведення даних у текстовий файл або передачу їх на логічний пристрій.

5. Процедура WriteLn ([Var f;] v1, v2, ..., vn) - забезпечує вивід інформації в текстовий файл або її передачу на логічний пристрій виводу. Процедура повністю ідентична процедурі Write, за винятком того, що виводиться рядок символів завершується символами # 13 і # 10. При виклику WriteLn допускається опускати список виведення, в цьому випадку в файл передається маркер кінця рядка (при виведенні на екран це призведе до переведення курсору на початок наступного рядка).

6. Функція SeekEOLn ([Var f]): boolean - пропускає всі прогалини і знаки табуляції до маркера кінця рядка або до першого значущого символу і повертає TRUE при виявленні маркера. Якщо файлова змінна не вказана, то функція перевіряє стандартний файл INPUT.

7. Функція SeekEOF ([Var f]): boolean - пропускає всі прогалини, знаки табуляції і маркери кінця рядка до маркера кінця файлу або до першого значущого символу і повертає TRUE при виявленні маркера. Якщо файлова змінна відсутня, то функція перевіряє стандартний файл INPUT

8.Функція SetTextBuf - назначає для текстового буфер ввода-виводу.

9. Функція Flush сбрасує для текстового файлу буфер вводу.

10. Функція Append відкриває існуючий фай для додання елементів в нього.

Нетипізовай файли:

file F: file;

Нетипізований називають файли, оголошені без вказівки типу його компонентів. Операції читання і запису з такими файлами здійснюються блоками. Відсутність типу компонента робить ці файли сумісними з будь-якими іншими, а виконання вводу / виводу блоками дозволяє організувати високошвидкісний обмін даними між диском і пам'яттю. Нетипізовані файли, як і типізовані, допускають організацію прямого доступу. Для визначення в програмі нетипізованого файлу служить зарезервоване слово File:

Нетипізовані файли - це послідовність байтів;

Відмінність від типізованих файлів є в тому, що нетипізований файл не має жестко встановленої одиниці зчитування/запис, як типізовані файли. В нетипізованих файлах за одне звернення зчитаються/записується число байтів приблизно = величині буфера вводу/виводу, що сприяє збільшенню швидкості роботи з файлами.

Процедури:

1. Ті самі що й в звичайних файлах, виключення становлять Read i Write, за мість них BlockRead, BlockWrite. Процедури Reset i Rewrite мають 2й параметр типу Word, який визначає розмір запису, використовуваний при передачі данних. Якщо цього параметра не має, розмір запису приймається по замовчуванню рівним в 128 байтів.
2. Процедурні типи, їх використання.

Процедурні типи дозволяють оголосити змінні, яким допускається використовувати імена процедур або функцій, а також передавати такі процедури та функції як параметри.

Type <ідентифікатор типу> = procedure [(список формальних параметрів)];

<ідентифікатор типу> = function [(список формальних параметрів)] : <тип результату>;

Приклад:

Type Proc = procedure (x, y : real; var i : integer);

Func = function(x : real) : real;

Non = procedure;

В процедурному типі допускаються вкладені виклики:

Type ErrorFun = function (case : integer) : Boolean;

FileFun = function (ind : longint; F : ArrayFun) : byte;

Функція не може повертати значення процедурного типу.

Для використання присвоювання змінній процедурного типу процедури потрібно дотримуватися сумісності по присвоюванню у виразах:

1. вони повинні мати однакову кількість параметрів;
2. параметри повинні мати однаковий тип при присвоюванні процедури або функції;
3. процедура або функція не повинна бути стандартною або визначеною в якомусь з модулів;
4. така процедура не повинна бути вкладеною;

Типізована константа процедурного типу може бути описана таким чином: для початку описується процедурний тип, далій описуємо саму процедуру цього типу, а потім вже описуємо саму константу. Наприклад:

type MyFunc = function (x, n : integer) : longint;

function Step(x, n : integer): longint; far;

begin

step:=exp(n\*ln(x));

end;

const Step1 : MyFunc = Step;

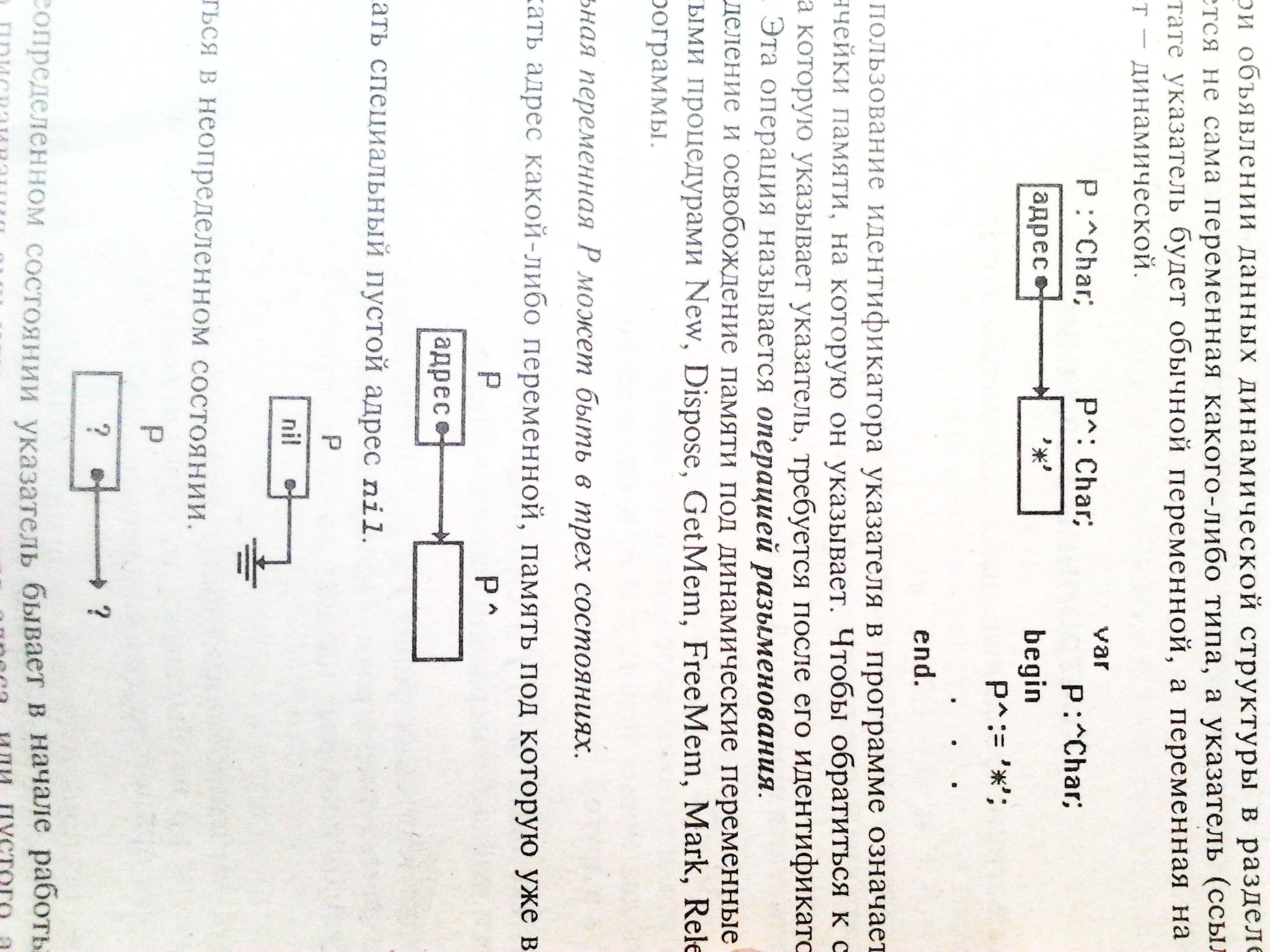
Змінна процедурного типу є адресою пам’яті де розміщена процедура або функція.

Процедурний тип доцільно використовувати, коли в програмі використовують процедури чи функції з подібним інтерфейсом.

Процедури або функції, ідентифікатори яких будуть передаватися в якості параметрів процедурного типу, по правилам мови необхідно компілювати в режимі дальнього виклику (з вказування директиви компілятора {$F+} або службового слова far). В цьому режимі при викликові підпрограм використовується довгі 4-байтові адреси на відміну від коротких 2-байтових, які використовуються для адресації програм, оголошених в основній програмі чи її підпрограмі. При використанні відкриваючої директиви компілятора {$F+} потрібно використовувати також закриваючу директиву {$F-}.

1. **Вказівники. Константа nil. Породження та знищення динамічних об’єктів. Дії над вказівниками. Стандартні процедури та функції для роботи з вказівниками.**

При оголошенні даних динамічної структури в розділі описань вказується не сама змінна певного типу, а вказівник (посилання) на неї. А результаті вказівник буде звичайною змінною, а змінна на яку він вказує - динамічною.



Використання ідентифікатора вказівника в програмі означає звернення до адреси комірки в пам’яті, на яку він вказує. Щоб звернутися до даних комірки пам’яті, на яку вказує вказівник, потріно після ідентифікатора поставити символ ^. Ця операція називається операцією розйменування.

Виділення та звільнення пам’яті під динамічні змінні виконується стандартними процедурами New, Dispose, GetMem, FreeMem, Mark, Release під час роботи програми.

Вказівник змінної може бути в трьох станах:

1. Містити адресу будь-якої змінної, пам’ять під яку вже виділена.
2. Містити спеціальну пусту адресу nil.
3. Знаходитися в невизначеному стані.

В невизначеному стані вказівник буває на початку роботи програми до першого присвоювання йому або конкретного адресу, або пустого значення nil, а також після звільнення комірки пам’яті на яку він вказував.

Відмінність між невизначеним станом вказівників і станом nik, в тому що, в стані nil при порівняння вказвники будуть однаковими, а в невизначеному стані ймовірність того що при порівнянні вказівники будуть однаковими є майже неможливою, але реальною.

Процедура New(P) виділяє область пам’яті відповідного тому типу, який описаний для вказівника Р і записує адрес виділеної пам’яті в вказівник.

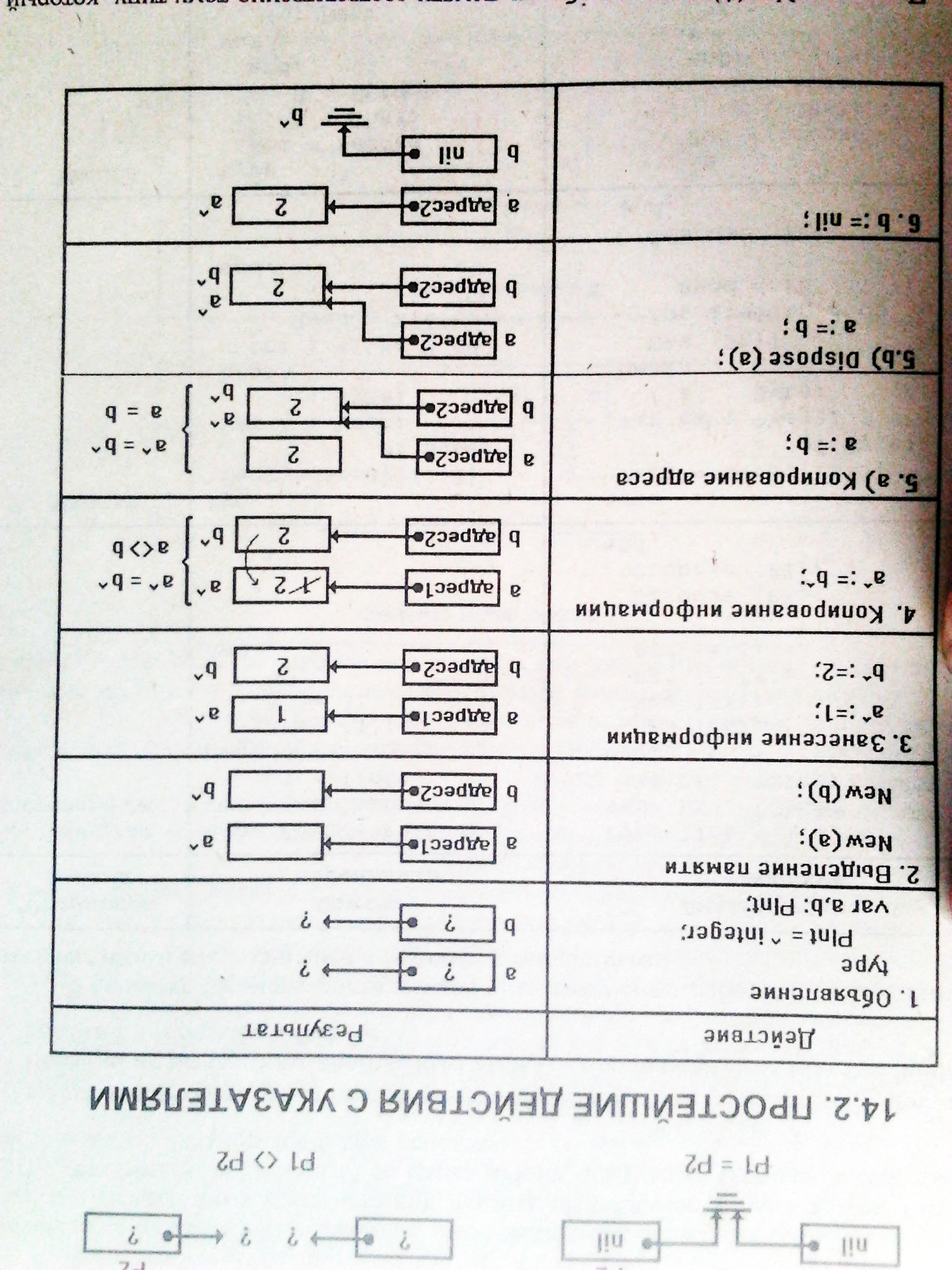
Процедура Dispose(P) звільняє область пам’яті, на яку вказує вказіник P, після чого ця область пам’яті стає доступною для розподілення під інші динамічні змінні.

Процедура GetMem(P,20) виділяє область пам’яті відповідного тому типу, який описаний для вказівника Р розміром 20 байт, і записує адрес виділеної пам’яті в вказівник.

Процедура FreeMem(P,20) звільняє область пам’яті, на яку вказує вказіник P розміром 20 байт, після чого ця область пам’яті стає доступною для розподілення під інші динамічні змінні.

Процедура Mark(P) записує стан кучі в вказівник Р.

Процедура Release(P) повертає кучу в заданий стан, тобто звільняє всю пам’ять яка не міститься в кучі.



1. **Базові поняття та визначення ООП. Порядок створення об’єктів та екземплярів. Властивості: інкапсуляція, наслідування, поліморфізм. Область дії полів та методів об’єкту. Переваги ООП.**

ООП - це методологія програмування, яка основана на представлені програми в вигляді совокупності об`єктів, кожен з яких є реалізацією певного класу, а класи утворюють ієрархію на принципах успадкування.

Область дій полів об’єктів - неявно розповсюджується на методи даного об’єкту, тому формальні параметри метода не можуть співпадати з іменем полем даного об’єкта. Звернення до власних змінних об’єкта досягається шляхом неявної передачі параметра self, який указує на те чи належить змінна тому ж об’єкту що і метод .

Procedure Person.Inp;

begin

write(‘name’);

readln(self.main);

writeln(‘surname?’);

readln(sel.surname);

end.

Явно параметр self указується тоді коли за межами обєкта існують змінні з такими ж індекаторами, свої власті зовнішні описання.

**Основні принципи об'єктно-орієнтованого програмування.**

**Інкапсуляція**

Інкапсуляція - це механізм, який об'єднує дані і код, який маніпулює цими даними, а також захищає і те, і інше від зовнішнього втручання або неправильного використання, тобто, інкапсуляція означає, що методи (коди) і дані одночасно представлені в одній і тій же структурі.

**Успадковування**

Це процес породження нових об'єктів-нащадків від існуючих об'єктів-батьків, при цьому нащадок успадковує від батька всі його поля, властивості і методи, які надалі можна використовувати в незміненому вигляді або перевизначати (модифікувати).

Спадкування дозволяє різним типам даних спільно використовувати один і той же код, приводячи до зменшення його розміру та підвищенню функціональності.

**Поліморфізм -** це властивість, яка дозволяє одне і те ж ім'я використовувати для вирішення двох або більше схожих, але технічно різних завдань.

Метою поліморфізму, стосовно об'єктно-орієнтованого програмування, є використання одного імені для завдання загальних для класу дій. Виконання кожного конкретного дії буде визначатися типом даних.

1. **Статичні та віртуальні методи. Конструктори та деструктори.**

Адреса які зв’язані з екземплярами об’єктів (змінних об’єктного типу) статично ще під час компіляції і не змінюється під час завершення роботи програми, називається статичним методом.

Методи можуть бути статичними (static - за замовчуванням), віртуальними (virtual) або динамічними (dynamic). Віртуальні та динамічні методи можуть бути перекритими (override) і абстрактними (abstract). Ці специфікатори (designators) методів грають роль тоді, коли об'єктом є клас-спадкоємець. Вони визначають, яка саме реалізація методу буде викликана в тому чи іншому випадку.

Статичні методи (Static methods).

Методи є статичними за замовчуванням і слово static не вказується. Коли викликається статичний метод, для визначення конкретної реалізації методу використовується інформація часу компіляції. У наступних описах методи Draw є статичними:

type

TFigure = class

procedure Draw;

end;

TRectangle = class (TFigure)

procedure Draw;

end;

Віртуальні та динамічні методи.

Для того щоб зробити метод віртуальним або динамічним, треба додати відповідну директиву в його опис. Ці методи, на відміну від статичних, можуть бути перекритими (override) в класах спадкоємців.

Коли викликається перекритий метод, дійсний, тобто часу виконання, тип класу або об'єкта, використаний у виклику методу (а не оголошений тип), визначає, яка реалізація буде викликана.

Для перекриття методу його треба оголосити до директиви override. Тема перекриває методу повинен в точності відповідати заголовку методу в батьківському класі. У програмі Method\_Binding ілюструється специфіка віртуальних і перекритих методів.

Незважаючи на те, що тільки віртуальні та динамічні методи можуть перекриватися, всі методи можуть бути перевантажені (overload).

Віртуальні та динамічні методи є семантично еквівалентними. Різниця між цими методами виявляється тільки на етапі виконання в способі реалізації власне виклику методу. Віртуальні методи оптимізовані за швидкістю виклику, в той час як динамічні - за обсягом коду.

В основному, віртуальні методи є найбільш ефективним шляхом досягнення переваг поліморфізму. Динамічні методи корисні тоді, коли клас має багато потенційно перекриваються методів, які успадковуються численними похідними класами додатки, проте тільки зрідка насправді перекриваються.

Конструктор - є спеціальним типом процедури, яка виконує початкові установки для роботи механізму віртуальних методів.

Перед викликом будь-якого віртуального методу якогого-небудь об’єкта необхідно викликати конструктор цього об’єкта.

Кожен окремий екземпляр об’єкта повинен бути ініціалізованим окремим викликом конструктора.

Деструктор - так само як і констуктор, спеціальний вид методу, який по іньшому називають “збирач сміття”.

Деструктори є протилежностями конструкторів і використовуються для очищення об'єктів після їх користування. Зазвичай очищення складається з видалення всіх полів-вказівників в об'єкті.

1. **Динамічні об’єкти. Помилки, що виникають при роботі з динамічними об’єктами. Директиви public та private.**

Динамічні об’єкти - описуються так як і динамічні змінні тільки змінна динамічного об’єкту містить вказівник на об’єкт.

var <ідентифікатор вказівника на тип> : ^<тип об’єкту>

var strpointer : ^ Obj\_Array;

Звернення до

* об’єкта “в цілому”: strpointer^
* полів об’єкту: strpointer^.<ім’я поля>, strpointer^.name
* методів об’єкту: strpointer^.<ім’я методу>, strpointer^.print

**Обробка помилок що виникають при роботі з динамічними об’єктами**

Якщо перевизначити функцію heaptunc, то в разі помилки вказівник буде мати значення nil, а конструктор не буде виконуватися. Але конструктор сам може містити дії по розміщенні полів в динамічні області пам’яті. Якщо в цьому випадку не вистачить пам’яті то логічним буде відмінити всі зроблені розподілення і звільнити пам’ять та надати вказівнику значення nil.

Процедура Fail надає вказівнику значенні nil, вона може бути використана конструктором та записується без параметрів.

Призначення Fail:

* викликає динамічний екземпляр який було розміщено в пам’яті до входження в конструктор і повертає йому значення nil.

Крім того саме ім’я конструктора можна використовувати як логічну функцію, яка повертає false якщо в його тілі була викликана процедура fail, а в іншому випадку true.

Деректива Public - оголошує загальнодоступні поля та методи, вона діє за замовчуванням.

Директива Private - оголошує поля і методи не доступними для оголошення ззовні.

type

<ідентифікатор> = object

Public

<загальноступні поля>

<загальноступні методи>

Private

<приховані поля>

<приховані методи>

end;

В даному прикладі директива Public може бути відсутнім, так як вона є директивою за замовчуванням.

1. **Концепція RAD. Система розробки програм Delphi та її IDE.**

**Швидка́ розро́бка застосу́нків**, **RAD** (від [англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *rapid application development*) — концепція створення засобів розробки [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA), [програмних продуктів](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82), що приділяє особливу увагу швидкості й зручності[програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), створенню технологічного процесу, що дозволяє програмістові максимально швидко створювати [комп'ютерні програми](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0). З кінця [XX століття](https://uk.wikipedia.org/wiki/XX_%D1%81%D1%82%D0%BE%D0%BB%D1%96%D1%82%D1%82%D1%8F) RAD одержала широке поширення й схвалення. Концепцію **RAD** також часто зв'язують із концепцією [візуального програмування](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B7%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

## Основні принципи RAD

* Інструментарій має бути націлений на мінімізацію часу розробки.
* Створення прототипу для уточнення вимог замовника.
* Циклічність розробки: кожна нова версія продукту грунтується на оцінці результату роботи попередньої версії замовником.
* Мінімізація часу розробки версії, за рахунок перенесення вже готових модулів і додавання функціональності в нову версію.
* Команда розробників повинна тісно співпрацювати, кожен учасник повинен бути готовий виконувати декілька обов'язків.
* Управління проектом повинне мінімізувати тривалість циклу розробки.

## Фази розробки

1. Планування (Requirements Planning phase) - сукупність вимог, отриманих при системному плануванні та аналізі процедури розробки життєвого циклу (SDLC). На цьому етапі користувачі, менеджери та IT-фахівці обговорюють завдання проекту, його обсяг, системні вимоги, а також складнощі, які можуть виникнути при розробці. Фаза завершується узгодженням ключових моментів з RAD-групою і отриманням від керівників проекту дозволу на продовження.
2. Користувальницьке проектування (User design phase) - протягом даного етапу користувачі, взаємодіючи з системними аналітиками, розробляють моделі і прототипи, які включають в себе всі необхідні системні функції. Для перекладу користувацьких прототипів в робочі моделі RAD-група зазвичай використовує техніку об'єднаної розробки додатків (JAD) і CASE-інструменти. Користувальницьке проектування виявляється тривалим інтерактивним процесом, який дозволяє користувачам зрозуміти, змінити і в кінцевому рахунку вибрати робочу модель, що відповідає їх вимогам.
3. Конструювання (Construction phase) - етап, в якому основна задача полягає в розробці програм і додатків. Аналогічна стадії "реалізація" в SDLC. У RAD, однак, користувачі продовжують брати участь і як і раніше можуть пропонувати зміни або поліпшення у вигляді розроблених ними доповідей. В їх завдання входить програмування і розробка додатків, написання коду, інтеграція модулів і системне тестування.
4. Перемикання (Cutover phase ) - включає в себе операції по конверсії даних, тестування, перехід на нову систему і тренування користувачів. За своїм завданням нагадує фінальну стадію SDLC. Порівнюючи з традиційними методами розробки ПЗ, весь процес виявляється стислим за часом. Як результат, нова система виявляється швидше побудованої, доставленої до замовника і встановленої на робочих місцях.

**Delphi** (Де́лфи, произносится [/ˈdɛlˌfi:/](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82)[[1]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-1)) — [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [структурированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) со [строгой](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8_%D1%81%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [статической типизацией](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) переменных. Основная область использования — написание прикладного программного обеспечения.

Первоначально носил название [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) и исторически восходит к одноименному диалекту языка, разработанному в фирме [Apple](https://ru.wikipedia.org/wiki/Apple) в [1986 году](https://ru.wikipedia.org/wiki/1986_%D0%B3%D0%BE%D0%B4) группой [Ларри Теслера](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%80%D1%80%D0%B8_%D0%A2%D0%B5%D1%81%D0%BB%D0%B5%D1%80)[[2]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-2). Однако в настоящее время термин [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) чаще всего употребляется в значении языка [среды программирования Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)). Начиная с Delphi 7[[3]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-3), в официальных документах [Borland](https://ru.wikipedia.org/wiki/Borland) стала использовать название Delphi для обозначения языка [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal).

**Интегри́рованная среда́ разрабо́тки**, **IDE** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Integrated development environment*) — система программных средств, используемая [программистами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) для разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (ПО).

Среда разработки включает в себя:

* [текстовый редактор](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80),
* [компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) и/или [интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80),
* средства автоматизации сборки,
* [отладчик](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA).

## Целевая платформа

Изначально среда разработки Delphi была предназначена исключительно для разработки приложений [Windows](https://ru.wikipedia.org/wiki/Windows), затем был реализован вариант для платформ [Linux](https://ru.wikipedia.org/wiki/Linux) (как [Kylix](https://ru.wikipedia.org/wiki/Kylix)), однако после выпуска в [2002 году](https://ru.wikipedia.org/wiki/2002_%D0%B3%D0%BE%D0%B4) Kylix 3 его разработка была прекращена, и вскоре было объявлено о поддержке [Microsoft .NET](https://ru.wikipedia.org/wiki/.NET_Framework), которая, в свою очередь, была прекращена с выходом Delphi 2007.

На сегодняшний день, наряду с поддержкой разработки 32 и 64-разрядных программ для Windows, реализована возможность создавать приложения для Apple [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X_10.0) (начиная с Embarcadero Delphi XE2), [iOS](https://ru.wikipedia.org/wiki/IOS) (включая симулятор, начиная с XE4 посредством собственного компилятора), а также, в Delphi XE5, для Google Android (непосредственно исполняемые на [ARM](https://ru.wikipedia.org/wiki/ARM_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0))-процессоре).

Независимая, сторонняя реализация среды разработки проектом [Lazarus](https://ru.wikipedia.org/wiki/Lazarus) ([Free Pascal](https://ru.wikipedia.org/wiki/Free_Pascal), компиляция в режиме совместимости с Delphi) позволяет использовать его для создания приложений на Delphi для таких платформ, как Linux, [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X) и [Windows CE](https://ru.wikipedia.org/wiki/Windows_CE).

Также предпринимались попытки использования языка в проектах [GNU](https://ru.wikipedia.org/wiki/GNU) (например, [Notepad GNU](https://ru.wikipedia.org/wiki/Notepad_GNU)) и написания компилятора[[*источник не указан 2153 дня*]](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B8_%D0%BD%D0%B0_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8) для [GCC](https://ru.wikipedia.org/wiki/GCC).

## Философия и отличия от популярных прикладных языков программирования

При создании языка (и здесь качественное отличие от яз ыка C) не ставилась задача обеспечить максимальную производительность исполняемого кода или лаконичность исходного кода для экономии оперативной памяти. Изначально, язык ставил во главу угла стройность и высокую читаемость, поскольку был предназначен для обучения дисциплине программирования. Эта изначальная стройность, в дальнейшем, как по мере роста аппаратных мощностей, так и в результате появления новых парадигм, упростила расширение языка новыми конструкциями.

Так, сложность объектного C++, по сравнению с C, выросла весьма существенно и затруднила его изучение в качестве первого языка программирования, чего нельзя сказать об Object Pascal относительно Pascal.

1. **Візуальна технологія розробки програм.**

**Візуа́льне програмува́ння** — спосіб створення програм шляхом маніпулювання графічними об'єктами замість написання [програмного коду](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) в [текстовому](http://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82) вигляді.

Візуальне програмування дозволяє програмувати, використовуючи графічні або символьні елементи, якими можна маніпулювати інтерактивним чином згідно з деякими правилами, причому просторове графічних об’єктів використовувати в якості елементів синтаксису програми. Значна частина візуальних мов програмування базується на ідеї «фігур і ліній», де фігури ([прямокутники](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BC%D0%BE%D0%BA%D1%83%D1%82%D0%BD%D0%B8%D0%BA), овали та ін.) розглядаються як суб'єкти і з'єднуються лініями (стрілками, дугами тощо), які являють собою відношення. Приклад: [UML](http://uk.wikipedia.org/wiki/UML)

Мови візуального програмування можуть бути додатково класифіковані в залежності від типу і ступеня візуального вираження, на типи[[*Джерело?*](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BA%D1%96%D0%BF%D0%B5%D0%B4%D1%96%D1%8F:%D0%9F%D0%BE%D1%81%D0%B8%D0%BB%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BD%D0%B0_%D0%B4%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0)]:

* Природно-візуальні мови мають невід'ємне візуальне вираження, для якого немає очевидного текстового еквіваленту (наприклад, графічна мова G в середовищі [LabVIEW](http://uk.wikipedia.org/wiki/LabVIEW)).
* Візуально-перетворені мови є невізуальними мовами з накладеним візуальним представленням.

Значна кількість сучасних мов програмування має розвинуті візуальні засоби для розробки графічного інтерфейсу, причому здійснюється програмування розміщених на спеціальних формах об'єктів з настроюванням їх властивостей та поведінки. [CodeGear Delphi](http://uk.wikipedia.org/wiki/CodeGear_Delphi) і [C++ Builder](http://uk.wikipedia.org/w/index.php?title=C%2B%2B_Builder&action=edit&redlink=1), [Microsoft Visual Studio](http://uk.wikipedia.org/wiki/Microsoft_Visual_Studio) та мови, які включає в себе цей засіб ([Visual Basic](http://uk.wikipedia.org/wiki/Visual_Basic), Visual C#, Visual J# тощо) часто плутають з візуальними мовами програмування. Всі ці мови є текстовими, а не візуальними (графічними). MS Visual Studio та Delphi є [візуальними середовищами програмування](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B7%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), але не візуальними мовами програмування.

1. **Подіє-орієнтоване програмування.**

**Подійно-орієнтоване програмування** ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *event-driven programming*; надалі ПОП) - парадигма програмування, в якій виконання програми визначається подіями - діями користувача (клавіатура, миша), повідомленнями інших програм і потоків, подіями операційної системи (наприклад, надходженням мережевого пакета).

ПОП можна також визначити як спосіб побудови комп'ютерної [програми](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), при якому в коді (як правило, в головний функції програми) явним чином виділяється*головний цикл програми*, тіло якого складається з двох частин:*отримання повідомлення про* [*подію*](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%96%D1%8F#.D0.86.D0.BD.D1.84.D0.BE.D1.80.D0.BC.D0.B0.D1.82.D0.B8.D0.BA.D0.B0) і *обробки події*.

Як правило, в реальних завданнях виявляється неприпустимим тривале виконання обробника події, оскільки при цьому програма не може реагувати на інші події. У зв'язку з цим при написанні подійно-орієнтованих програм часто застосовують [автоматне програмування](http://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

## Сфера застосування

Подійно-орієнтоване програмування, як правило, застосовується в трьох випадках:

1. При побудові користувацьких інтерфейсів (в тому числі [графічних](http://uk.wikipedia.org/wiki/GUI));
2. При створенні серверних застосунків у разі, якщо з тих чи інших причин небажано породження обслуговуючих [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0));
3. При програмуванні ігор, в яких здійснюється управління значною кількістю об'єктів.
4. **Мова програмування Delphi (Object Pascal).**

**Delphi** (Де́лфи, произносится [/ˈdɛlˌfi:/](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82)[[1]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-1)) — [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [структурированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) со [строгой](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8_%D1%81%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [статической типизацией](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) переменных. Основная область использования — написание прикладного программного обеспечения.

Первоначально носил название [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) и исторически восходит к одноименному диалекту языка, разработанному в фирме [Apple](https://ru.wikipedia.org/wiki/Apple) в [1986 году](https://ru.wikipedia.org/wiki/1986_%D0%B3%D0%BE%D0%B4) группой [Ларри Теслера](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%80%D1%80%D0%B8_%D0%A2%D0%B5%D1%81%D0%BB%D0%B5%D1%80)[[2]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-2). Однако в настоящее время термин [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) чаще всего употребляется в значении языка [среды программирования Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)). Начиная с Delphi 7[[3]](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-3), в официальных документах [Borland](https://ru.wikipedia.org/wiki/Borland) стала использовать название Delphi для обозначения языка [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal).

**Интегри́рованная среда́ разрабо́тки**, **IDE** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Integrated development environment*) — система программных средств, используемая [программистами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) для разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (ПО).

Среда разработки включает в себя:

* [текстовый редактор](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80),
* [компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) и/или [интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80),
* средства автоматизации сборки,
* [отладчик](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA).

## Целевая платформа

Изначально среда разработки Delphi была предназначена исключительно для разработки приложений [Windows](https://ru.wikipedia.org/wiki/Windows), затем был реализован вариант для платформ [Linux](https://ru.wikipedia.org/wiki/Linux) (как [Kylix](https://ru.wikipedia.org/wiki/Kylix)), однако после выпуска в [2002 году](https://ru.wikipedia.org/wiki/2002_%D0%B3%D0%BE%D0%B4) Kylix 3 его разработка была прекращена, и вскоре было объявлено о поддержке [Microsoft .NET](https://ru.wikipedia.org/wiki/.NET_Framework), которая, в свою очередь, была прекращена с выходом Delphi 2007.

На сегодняшний день, наряду с поддержкой разработки 32 и 64-разрядных программ для Windows, реализована возможность создавать приложения для Apple [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X_10.0) (начиная с Embarcadero Delphi XE2), [iOS](https://ru.wikipedia.org/wiki/IOS) (включая симулятор, начиная с XE4 посредством собственного компилятора), а также, в Delphi XE5, для Google Android (непосредственно исполняемые на [ARM](https://ru.wikipedia.org/wiki/ARM_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0))-процессоре).

Независимая, сторонняя реализация среды разработки проектом [Lazarus](https://ru.wikipedia.org/wiki/Lazarus) ([Free Pascal](https://ru.wikipedia.org/wiki/Free_Pascal), компиляция в режиме совместимости с Delphi) позволяет использовать его для создания приложений на Delphi для таких платформ, как Linux, [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X) и [Windows CE](https://ru.wikipedia.org/wiki/Windows_CE).

Также предпринимались попытки использования языка в проектах [GNU](https://ru.wikipedia.org/wiki/GNU) (например, [Notepad GNU](https://ru.wikipedia.org/wiki/Notepad_GNU)) и написания компилятора[[*источник не указан 2153 дня*]](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B8_%D0%BD%D0%B0_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8) для [GCC](https://ru.wikipedia.org/wiki/GCC).

1. **Основні відмінності мови Delphi від Turbo Pascal.**

Первая версия появилась в мае 1995 г., когда еще не существовала Windows 95 (но тем не менее существовала Windows NT). Это единственная версия, работающая под управлением 16-разрядной Windows 3.1 (3.11). В ней впервые была опробована новая модель объектов, позаимствованная из различных объектно-ориентированных языков, и главным образом, из языка C++. Эта модель оказалась настолько революционной, что существовавшие в то время в поздних версиях Turbo Pascal *объекты* стали не нужны (их возможности полностью поглотила новая модель), а сама новая модель получила название *классов.*

Классы активно используют динамическую память, в связи с чем несколько изменилась нотация языка, а сам язык был назван Object Pascal. По сравнению с Turbo Pascal в него были внесены существенные дополнения и изменения, в том числе (здесь и далее перечисляются лишь наиболее важные, с моей точки зрения, изменения):

* введены открытые массивы и их конструкторы для передачи в подпрограммы массивов переменной длины;
* введена внутренняя для функции переменная Result и разрешено игнорировать возвращаемый функцией результат;
* сняты ограничения на тип возвращаемого функцией результата (этот тип может быть любым, за исключением объектов старого стиля и файлов);
* введен механизм обработки исключительных ситуаций.

Библиотека компонентов 1-й версии с самого начала показала основную направленность вновь разработанной системы: программирование баз данных. С этой целью первая и все последующие версии Delphi снабжаются специальным инструментом доступа к данным -BDE (Borland Database Engine - машина баз данных корпорации Borland), а также сервером баз данных InterBase (этот сервер производится филиалом Borland - компанией InterBase Software Corporation) и соответствующими средствами конфигурации сервера, его контроля и связи с ним.

Палитра компонентов первой версии состоит из 9 страниц и содержит 79 компонентов. В качестве дополнительных утилит поставлялись система генерации отчетов ReportSmith v.2.5 и входящая в ее состав автономно запускаемая утилита ReportSmith Run Time Viewer, утилита доступа к данным DatabaseDesktop v.5.1 и утилита конфигурации BDE - Database Engine Configuration v.2.5. Кроме того, утилиты WinSighte и WinSpecter предназначались для упрощения отладки разрабатываемых программ в среде 16-разрядных ОС Windows 3-х.