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

Мовою прог. можна назвати будь-яку систему позначень і понять для описання структур даних і алгоритм. Мова прог.-це множина речень і рядків визначена деякому кінцевому результату.

МОВИ ПРОГРАМУВАННЯ:

1. Fortran-1мова прог. високого рівня(1954р.);
2. Cobol; 3)ALGOL; 4)PL/1; 5)BASIC; 6)PASCAL(1968-1969p.); 7)ADA; 8)C(..70 Деніс Регі) -> C++(..80 Бьярі Страуструп) і C# NET Framework.

Класифікація мов прог.:

По реалізованих принципах мови прог. ділять на:

1. Мови прог. низького рівня;
2. МП високого рівня;

1-константи мови близькі до масштабного коду;

2-константи мови близькі до поняття вихідного завдання.

По класу вирішення задач:

1. Спеціалізовані; 2) Універсальні.

**Підтримувані**[**парадигми програмування**](http://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B8_%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)

[Об'єктно-орієнтовані](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%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), [логічні](http://uk.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D1%96%D1%87%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), [функційні](http://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%B9%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), структурні…

^Імперативні мови прог. : программа складається з послідовності команд(Fortran, Algol, Pascal, C++…) ;

^Функціоналі(аплікативні). Програма являє собою послідовність ф-цій і виразів, які потрібно обчислити (Lips, Haskel, Miranda);

^Процедурні МП. Програма представляє послідовність процедур дія реалізації алгоритму розвязку задачі;

^Декларативні(логічні) МП. Програма складається з визначень відносин між об’єктами і поставленою метою(Prolog);

^Обєктивно орієнтовні МП. Дані преставляють собою об’єкти стандартних або користувальних класів, алгоритми в них реалізовані, як послідовність викликів методів цих об’єктів (C#, Delphi, ADA

Компілятор-транслятор: Переведення програм написання мовою високих рівнів в мову машин кодів які називаються трансляторами.

Транслятори: 1) Компілятори; 2)Інтерпретатори.

Вихідна МВР представляє собою ланцюжок символів серед множини припустимих символів роздільників (пробіл, ентер.;)які розбивають речення на окремі слова ці слова називаються лексемами.

Синтаксис аналізатор-перевіряє коректність записів речень програми згідно до синт. правил мови та переводить послідовність лексем у внутрішній код компіляторів.

Код компіляторів- порядок дій на виконання.

Генератор коду- переводить внутришній код в машинний в процесі роботи звертається до набору таблиць куди вміщуються постійно для трансляції всіх програмних інформацій та інформ. індивідуальна для кожної програми.

Типи даних- множина значень даних і сукупність операцій припустима для цих даних.

Етапи розвязку задачі на ЕОМ:

1. Постановка задачі- формалізація це процес конкретизації проблеми до кінцевого числа термінів.
2. Складання алгоритму-алгоритмізація це процес складання алгоритму, що розв’язує поставлену задачу, використовуючи визначення на попередньому етапі структури і закономірності.
3. Програмування - складання програми (15%) це реалізація алгоритму мовою програмування.
4. Відладка програми- пошук та виправлення синтаксичних помилок.
5. Тестування програми-пошук та виправлення семантичних помилок.
6. Документування- оформлення документу.

**2.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.**

Алфавіт – це сукупність дозволених в мові символів чи груп символів, що розглядаються як єдине ціле. Алфавіт мови Паскаль складається з літер, цифр, спеціальних символів та символів, що не використовуються. Крім того до алфавіту мови Паскаль можна віднести зарезервовані слова та стандартні директиви . ***Спеціальні символи*** можна умовно розділити на розділові знаки, знаки пунктуації, знаки операцій та зарезервовані слова. Розділові знаки використовуються для відділення один від одного елементів програми. У якості розділових знаків можна використовувати : • символ пробілу; • символ табуляції; • коментарі. **Коментарем** називається будь-яка послідовність символів, розміщена у фігурні скобки: { коментар } . Основне призначення коментарів – пояснення по тексту програми. До знаків пунктуації відносяться: ( ) (\* \*) [ ] { } “ , . : ; := .. ^ @ $ # . Знаками операцій являються: + - \* / = < > <> <= >= . **Зарезервованими** у мові Паскаль є такі слова: and asm array begin Case const constructor destructor div Do downto else end file To type unit until uses Var;while with xor ... Стандартні директиви та зарезервовані слова в програмі можуть використовуватися лише за своїм прямим призначенням, і не можуть бути використані у будь-якій іншій якості. **Ідентифікатори**(імена) використовують для позначення констант, типів, змінних, процедур, функцій, класів модулей, програм і т.і. Ідентифікатори у Паскалі можуть бути будь-якої довжини, але значущими є лише перші 63 символи. Ідентифікатор може складатися з літер латинського алфавіту, цифр та символів підкреслювання, але починатися може лише з букви або символу підкреслювання. Регістр символів значення не має. До складу ідентифікаторів не можна включати спеціальни символи, символи, що не використовуються та символи пробілу. В якості ідентифікаторів не можна використовувати стандартні директиви, зарезервовані слова та імена елементів, що входять до стандартної бібліотеки. Приклади допустимих ідентифікаторів: a, alpha, MyVar, \_beta, n123 .Приклади недопустимих ідентифікаторів: 5a – починається з цифри; block# - містить спеціальний символ # ; My Var – містить символ пробілу ; mod – зарезервоване слово.

**Структура програми :** <програма>::=[< заголовок>];< блок> <заголовок>:: =program< імя\_програми> <імя\_програми>::=< ідентифікатор> <блок>::=[<розділ\_модулів>]; [<розділ\_міток>]; ……………………………………………………………….. [<розділ\_операторів>];

**Розділ модулів:** uses- описується 1 раз після заголовку <розділ\_модулів>::=<пусто>/uses<імя\_модуля>{,<імя\_модуля>} uses crt, graph, print. Розділ модулів описується тоді коли в програмі використовується: типи, змінні, константи, ф-ції, процедури…

**Розділ міток:** <розділ міток>::=<пусто>/label<мітки>{<мітки>} <мітки>::<ідентифікатор>/<цілі\_без\_знака> ,де ідентифікатор - символ мітка, а цілі\_числа\_знака – числова мітка від 9999….0. Розділ описується тоді коли є необхідним використовувати операцію goto.

**Розділ констант:** рі <розділ\_констант>::=<пусто>/const <описання\_коннстанти>{<опис\_константи>} <описання\_константи>::=<імя\_константи>=<значення\_константи> <імя\_константи>::=<ідентифікатор> <значення\_константи>::=<число без знаку>/<число зі знаком>/<рядок>/<константа>/ <константа зі знаком>/<константа виразу> <знак>:=+/- <рядок>::=<символ>{символ} <символ>::=<будь-який вираз алфавіту> constm=100; l=-m; n=10000; p='turbo'; k=m+n. Константа вираз-це вираз операндами якого є константи або функції від констант. 1) Можна використовувати цілі десяткові числа; 2)Цілі шіснадцяткові: $0; $48;  
3)Дійсні десяткові: форма фіксованою комою і форма з послідовною комою. Константи: 1)прості:const, length=1, min=-100, max=maxint, t=true, char1=’a’,str=’kpi’; 2)типізовані: const, dlina: intever=10, ch1: cror=q;

**Розділ типів:** Тип визначає множину даних значень і сукупність операцій припустимих для цих значень. <розділ типів>::=<пусто/type<описання\_типу>{;<описання\_типу>} <опис\_типу>::=<імя типу>=<тип> <імя типу>::=<ідентифікатор> Типи:1)прості(скалярні):а)Стандартні:цілі, дійсні, булевський, символьний, вказівник, рядковий. б) користувацькі: обмежуваний(тип діапазон), перелічувальний. 2) Складні:масив, множина, об’єкт, файл, процедурний. Порядковий тип – це типи у яких означення представляють собою впорядковану послідовність де кожен елемент має свій порядковий номер ціле число.

**Розділ змінних:** <розділ змінних>::<пусто>/var<описання змінної>{;<описання змінної>} <описання змінної>::=<ідентифікатор>{;<ідентифікатор>}:<типи> Типи: стандартні, короткі.

**Розділ процедур та функцій:** Підпрограми: стандартні(використовуються без попереднього вписання), визначені користувачем(описання в розділі описань процедури функції). Procedure=<імя процедури>; <розділ описання процедури > begin <розділ операторів процедури> end.

Function=<імя функції >; <розділ описання функції > begin <розділ операторів функції> end.

**Розділ операторів:** <розділ>::=begin< операторів список операторів>;end.

**3. Типізовані константи: описання та використання. Типізовані константи стандартних типів. Типізовані константи-множини. Типізовані константи-масиви. Типізовані константи-записи** Типізована константа визначається наступним чином : <константа>:< тип>=< константний вираз>; Наприклад: type color(blue, read, yellow); const pi:real=3.14; Значення констант обчислюються при компіляції і, не можуть бути змінені під час виконання програми. Вийнятком є типізовані константи, значення яких може бути змінено під час виконання програми, за умови, що була виконана 21 директива компілятора {$J+}. Якщо ж була виконана директива {$J-}, то типізовані константи перетворюються на іменовані константи, і їх значення не можна змінити під час виконання програми. Якщо в програмі не встановлено якусь із цих директив явним чином, то діє директива {$J+}. Типізовані константи неможна використовувати при описанні інших констант або типів. Типізовані константи не може бути файлового типу. Типізовані константи неможна використовувати при описані типу діапазону. Ініціалізація типіз. конст. відбувається лише 1 разі при повторному вході в блок зберігає попереднє своє значення.

Тип.конст.масив: const X:array[1..7] of bute=(2,8,6,5,0,1,9);

Y:array[1..2,1..5 ] of char=((‘a’,’b’,’c’,’d’,’e’)(‘f’,’g’,’k’,’I’,’j’))

Елементи, що розташовані в внутрішніх дужках відповідають останій розмірності масиву.

Тип.конст. множина : type letters=set of ‘d’..’n’;

Digits =set of 0..9;

Const set1:letters=[‘k’,’l’,’m’];

Set2: digits=[2..4]

Set3:set of char=[‘a’..’z’];

Кожний компонент константи множини може представляти собою окрему дану відповідного типу, або інтервал значень,що складається з двох констант.

Тип.конст-запис

Type student=record

PIB: string;

Kurs:byte;

Vik:integer:

Stat:char;

End;

Grupa=array[1..3] of student;

Const

Stud1:student=(PIB’Tarasenko O.V.’,kurs……..);

Grupa KV:Grupa=(PIB:’….’)

В описанні тип.конст. запису указуються значення усіх полів а також їх ідентифікатор. При описанні тип.конст. запису не допускаються поля файлового типу. Поля перераховуються в тому ж порядку як і при описанні типу.***2 семестр***

1. Мови програмування. Класифікація мов програмування. Спрощена модель компілятора. Етапи розв’язку задачі на ПЕОМ.
2. 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.
3. Типізовані константи: описання та використання. Типізовані константи стандартних типів. Типізовані константи-множини. Типізовані константи-масиви. Типізовані константи-записи. Типізовані константи процедурного типу. Приклади.
4. Розділ типів. Типи даних в ТурбоПаскалі. Стандартні та користувацькі типи даних. Приклади.
5. Операнди, вирази, операції. Класифікація операцій. Приорітет операцій. Приклади.
6. Введення і виведення даних, форматування введення-виведення. Форми подання чисел в ТурбоПаскалі. Приклади.
7. Розділ операторів. Оператор присвоювання, оператор процедури, оператор переходу, складений оператор. Приклади.
8. Умовний оператор. Вкладеність умовного оператора. Оператор вибору case. Особливості його використання. Порівняння роботи умовного оператора та оператора вибору. Приклади.
9. Оператори повтору. Порівняння роботи операторів повтору. Процедури, що змінюють хід роботи циклічного оператора. Приклади.
10. Масиви. Основні визначення. Описання масивів. Операції над масивами. Приклади.
11. Рядки. Різні способи подання рядків. Поняття довжини рядка. Операції над рядками. Приклади.
12. Записи. Фіксовані записи, їх описання в програмі. Операції над записами. Оператор приєднання with. Переваги його використання. Приклади.
13. Записи. Варіантні записи, їх описання в програмі. Приклади.
14. Процедури та функції. Описання процедур та функцій. Область дії ідентифікаторів. Глобальні та локальні імена. Приклади.
15. Процедури та функції. Передача параметрів: параметри-змінні, параметри-значення, параметри-константи, нетипізовані параметри, правила їх передачі.
16. Множини. Описання множин. Операції над множинами, їх приорітет. Приклади.
17. Рекурсія. Випереджаюче описання. Приклади.
18. Файли. Класифікація файлів. Операції над файлами. Стандартні процедури та функції для роботи з файлами. Відмінности файлів від інших структур даних.
19. Типізовані файли прямого доступу. Типізовані файлі послідовного доступу. Приклади.
20. Текстові файли. Нетипізовані файли. Приклади.
21. Процедурні типи, їх використання.
22. Вказівники. Константа nil. Породження та знищення динамічних об’єктів. Дії над вказівниками. Стандартні процедури та функції для роботи з вказівниками.
23. Базові поняття та визначення ООП. Порядок створення об’єктів та екземплярів. Властивості: інкапсуляція, наслідування, поліморфізм. Область дії полів та методів об’єкту. Переваги ООП.
24. Статичні та віртуальні методи. Конструктори та деструктори.
25. Динамічні об’єкти. Помилки, що виникають при роботі з динамічними об’єктами. Директиви public та private.
26. Концепція RAD. Система розробки програм Delphi та її IDE.
27. Візуальна технологія розробки програм.
28. Подіє-орієнтоване програмування.
29. Мова програмування Delphi (Object Pascal).
30. Основні відмінності мови Delphi від Turbo Pascal.

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

Механізм заміни формальних параметрів на фактичні дозволяє настроїти багаторазовий повторюваний алгоритм на конкретний варіант. Будь який форм.параметр може бути або пар-змінноє або пар-значення. Щоб визначити параметр як змінну в заголовку пишемо var. \_\_Якщо параметр описаний як пар-змінна, то при виклику процедури йому повинен відповідати фактичний параметр у вигляді змінної того ж типу.\_\_Якщо ж параметр описаний як пар-значення то при виклику йому може відповідати довільний вираз.\_\_Якщо параметр визначений як значення, то в момент виклику підпрограми фактичний параметричний вираз обчислюється і отриманий результат розміщується в тимчасово відведеній пам’яті(тільки на час роботи підпрограми).\_\_Якщо формальний параметр визначений як пар-змінна , то при виклику підпрограмі передається сама змінна а не її копія(тобто її адреса). Таким чином будь які зміни пар-значення в під-програмі ніяк не сприймаються викликаючою програмою, у той час як зміна пар-змінної приводить до зміни фактичного параметра викликаючою програмою.

Щоб визначити параметр як змінну в заголовку пишемо const. При такому описі передається адреса фактичного параметра, тому ми можемо тільки братии значення фактичного параметра, а змінювати його значення заборонено. Передвавати можемо всі типи крім файлових. Пар-константи слід використовувати у випадках , якщо потрібно передати структуру даних , яка займає великий об'єм пам'яті, але при цьому не можна змінювати початкові дані з алгоритмічної точки зору.

Отже, пар-змінні використовуються як засіб для передачі результату із під-програми в викликаючи програму, а пар-значення як засіб для передачі вхідної інформації в підпрограму. Однак, оскільки при описанні пар-значення відбувається копіювання у тимчасову пам’ять і якщо це буде масив великої розмірності то витрати часу і пам’яті на копіювання можуть стати визначальними для оголошення такого параметра пар-змінною.

**Білет№5 Операнди, вирази, операції. Класифікація операцій. Приорітет операцій.**

Вираз - це сукупність операндів з'єднаних знаками операцій і круглими дужками.

- унарні;

- бінарні;

Вираз Результат

-7 -7

-(-7) 7

6-3 3

false true

true or false true

2>=1 true

Операнди у виразі:

1. проста змінна
2. змінна з індексом
3. елемент запису
4. константа
5. ім'я функції
6. множина

Класифікація операцій

1. арфметичне
2. відношенняє
3. булевські
4. множинні
5. рядкові
6. взяття адреси

**1**. +, -, \*, /, div, mod

7div3=2

mod3=1

0div3=0

0mod3=0

Ці операції застосовуються до будь-яких даних:

**2.1**Арифметчне відношення

1<2→true

2=3→false

**2.2** Логічне порівняння

False<True→true

**2.3** Символьне порівняння

'a' < 'z'

**2.4** Порівняння рядків

1=3→False

4=4→True

False<true→True

'ABC'<''ABD'→true

'ABC'>'A'→true

**3.** Булевські операції

знач операндів notA A and B A or B

A B

T T F T T

T F F F T

F T T F T

F F T F F

**4.** Рядкові операції

Конкатенція (+)

var S1,S2:string

......

S1:='ntu';

S2:='kpi';

S1:=S1+S2; {S1='ntukpi'}

**5.** Операції над мноижанми:

1. Об'єднання(+)
2. Перетин(\*)
3. Різниця(-)
4. =,<>
5. є надмножиною(>=)
6. є підмножиною(<=)
7. in(належність)

[1,2,3]+[3,4,5,6][1,2,3,4,5,6]-множина

[1,2,3,4]-[3,4,5,6]=[1,2]

[1,2,3,4]\*[3,4,5,6]=[3,4]

[1,2,3]=[3,2,1] true - булеве значення

[1,2,3]<>[3,2,1] false

[1,2,3]<=[1,2,3,4] true

[1,2,3]>=[1,2,3,4] false

4 in [1,2,3] false

**6.** Операції взяття адреси

@

Ідентифікатори:

1)змінних

2)процедур

3)функцій

Пріорітет опеарцій

Всі операції виконуються зліва направо згідно їх пріоритету. Порядок виконання може бути змінений круглими дужками

Рівень Оперції Клас операцій

пріоритету

І @, not, -,+ Унарні

ІІ \*,/,div,mod, Бінарні типу

and,shl,shr множення

ІІІ +,-,or,xor Бінарні типу

конкатенація додавання

ІV <,>,=,<>,<=,>= Бінарні типу

in відношення

1. Операнди, що знаходяться за 2 операціями з різними пріоритетами зв'язуються з тією, що має більш висой пріоритет

2. Операнд, що знаходиться між 2 операціями зв'язується з тією, що знаходитсья ліворуч.

3. Вираз в дужках обчислуюється, як окремий операнд.

4. Операції з рівним пріоритетом, з можливим регулюванням плорядку дужками.

**Білет№4 Розділ типів. Типи даних в ТурбоПаскалі. Стандартні типи даних.** **. Користувацькі типи даних.**

Тип визначає множину значень даних і сукупність операцій припустимих для цих значень.

<розділ типів>::=<пусто>|type<опис типу>{;<опис типу>

<опис типу>::=<ім'я типу>=<тип>

<ім'я типу>::=<ідентифікатор>

**ТИП:**

**Прості(скалярні):**

1. **Стандартні:**

* цілий
* дійсний
* булевський
* символьний
* вказівний
* рядковий

1. **Користувацькі:**

* обмежений
* перелічувальний

**Складні:**

* масив
* множина
* об'єкт
* файл
* процедурний

Порядковий тип - тип у якому значення представляють собою вподяковану послідовність, де кожен елемент має свій порядковий номер.

***Стандартні***

**1.1 Цілі типи**

Тип Діапазон Пам'ять

Byte 0...255 1 байт

Shirtint -128...127 1 байт

Integer -32768...32767 2 байти

Word 0...65535 2 байти

Longint -2142483648... 4 байти

2142483647

**1.2 Дійсні типи**

Тип Діапазон Пам'ять

Real ±2.9E-39...1.7E38 6

Single ±1.5E-45...3.4E38 4

Double ±5.0E-324...1.7E308 8

Extended ±1.9E-4951... 10

1.1E4932

Comp -2E+63...2E+63-1

**1.3 Булевський**

Boolean

Butebool true 1

wordbool false 0

longbool

**1.4 Символьний**

char

const l:char='a';

znak:char='\*';

**1.5 Рядковий**

string 0...255 символів

const student: string='vasja petrov';

***користувацькі***

2.1 Діапазон

<діапазон>::=<конст1>..<конст2>

н-д: type den=1..31;

Litera='a'..'z';

Litera≠'z'..'a';

2.2 Перелічувальний

<перелічувальний>::=(<ім'я>{,<ім'я>})

type Season=(winter,summer,spring);

grupaKV=(student1,student2);

digiatals=('1','2','3');

До даних цього типу не можна застовсувати ні арифметичні операції, ні стандартні процедури вводу, виводу.

14.Процедури та функції. Область дії ідентифікаторів. Глобальні та локальні імена. Приклади.

Область дії ідентифікаторів зумовлюється місцем їх об’явлення. Всі ідентифікатори(імена) розділяються на локальні та глобальні. Локальні імена діють лише в межах блоку, де вони описані. А глобальні діють у всій програмі вцілому.

У випадку збігу імен у різних блоках локальне відміняє глобальне.

Значення глобальних імен є допустимим для вкладених блоків але не навпаки.

Приклад:

Program Primer;

Var x,y:integer;{глобальні}

Procedure H1;

Var x,y:real;{локальні}

Begin

X:=0;

Y:=0;

Writeln(x,y);

End;

Begin

X:=2;

Y:=1;

H1;

Writeln(x,y);

End.

Результат: 0,0

2,1

**6.** **Введення і виведення даних, форматування введення-виведення**.**Форми подіння чисел**

Введення даних в мові Паскаль відбувається за допомогою таких процедур :

1. Read(<список елементів вводу>);
2. Readln(<список елементів вводу>);

<елемент вводу>::=<проста змінна>|<змінна з індексом>|<елемент запису>.

Процедурами read та readln допускається лише зчитування даних цілого, дійсного, символьного та рядкового типу.

1. Введення значення в змінну символьного типу.

Процедура read зчитує один символ і присвоює його змінній.

Н-д: var A,B:char;

Begin

Read(a);

Read(b);

End;

При введенні з клавіатури ‘3’\_’7’

А буде дорівнювати ‘3’, а B - ‘7’.

2 .Введення значення в змінну цілого типу.

Процедура Read очікує надходження послідовності символів, що утворюють ціле число. Всі пробіли, знаки табуляції або признаки кінця строки пропускаються.

3.Введення значення в змінну рядкового типу(String).

Процедура Read зчитує стільки символів вихідних даних, скільки вказано в описі рядкової змінної. Пробіл при цьому, як розділювач не враховується. Процедура Read при зчитуванні рядкової змінної перехід на новий рядок не робить.

Відмінність роботи процедур Read та Readln заклечається в тому, що вона після дій процедури Read виконує перехід на новий рядок.

Виведення даних в мові Паскаль відбувається за допомогою таких процедур :

1. write(<список елементів виводу>);
2. Writeln(<список елементів виводу>);

<елемент виводу>::=<константа>|<змінна>|<змінна з індексом>|<елемент запису>|<вираз>.

Процедурами Write та Writeln допускається лише виведення даних цілого, дійсного, символьного, рядкового та булевського типу. Елемент виводу не може бути перелічувального типу.

Формат виводу з фіксованою точкою:

[пробіли] [-] <Цифри> [.<Знаки\_дроб.частини>]

Формат виводу з плаваючою точкою:

[-] <Цифра>.<Знаки\_дроб.частини> E [+!-<Показник\_степені>]

Процедура Writeln після виконання дій процедури write записує признак кінця строки

**Виведення цілих чисел**

Значення Вираз Результат

134 write(i) 134

287 write(i,i,i) 287,287,287

134 write(i:6) \_ \_ \_ 1 3 4

312 write(i+i:7) \_ \_ \_ \_ 6 2 4

**Виведення дійсних чисел**

Значення R Вираз Результат

715,432 write(R) \_ \_ 7.1543200000E+02

-1.919E+01 write(R) \_ -1.9190000000E+01

597.986 write(R/2) \_ \_ 2.8399300000E+02

511.04 write(R:15) 5.11040000E+02

-511.04 write(R:15) -5.11040000E+02

511.04 write(R:8:4) 511.0400

-46.78 write(R:7:2) -46.78

-46.76 write(R:9:4) 46.7600

**7.** **Оператори. Розділ операторів. оператор присвоювання, оператор процедури, оператор переходу ,складений оператор.**

Оператори призначені для опису дій, які будуть виконуватись при реалізації алгоритму.

Оператори відділяються один від одного символом «крапка з комою»(;). Крапка з комою не являється частиною оператора, це розділювач операторів. Тому після останнього оператора програми та після останнього оператора в поставному операторі, тобто перед ключовим словом end, крапку з комою ставити не обов’язково.

Інують такі оператори: 1)Прості оператори(не можуть в собі містити інших операторів): присвоювання, процедури переходу.

2)Структурні оператори: складений оператор, умовний, процедури, приєднання.

<розділ операторів>::=begin <опис операторів>; end.

**Н-д:**  begin

Write(‘Hi KV’);

End.

**До простих операторів належать:**

**1.Оператор присвоювання**

**<оператор присвоювання>**::=<ліва частина>:=<вираз>

<ліва частина>::=<проста змінна>|<змінна з індексом>|<кваліфіковане імя>

+ -

A:=B; a+1:=b;

Wet[i]:=max; Vect:=0;

Student\_name:=’Petrenko’ 2b:=1+2;

Вираз – це сукупність операндів, поєднаних знаком операцій або круглими дужками.

**2.Оператор процедури:**

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

Виконання оператора процедури призводить до активізації дій описаних в її тілі. Перед активізацією тіла процедури відбувається передача фактичних параметрів відповідним формальним параметрам.

Н-д: summa(a,m,m,sum);

Initialize;

Swap(x,y);

**3.Оператор переходу:**

**<оператор переходу>**::=goto<мітка>

Виконання оператора goto призводить до передачі управління на оператор,, перед яким стоїть указана в операторі goto мітка.

Оператор переходу протирічить принципам структурованого програмування, тому його використання не рекомендується.

**До структурних операторів належать:**

**1.Складений оператор.**

<Складений оператор>::=begin<оператор>{;<оператор>}end.

Структурні оператори включають в себе цілу групу операторів в єдине ціле, після чуго вони можуть раухватись одним оператором.Складений оператор складається з послідовності обєднуючих операторів, які розташовуються між ключовими словами begin та end.

**8.** **Оператори. Розділ операторів. Структурні оператори (; умовний оператор if та оператор вибору case).**

Оператори призначені для опису дій, які будуть виконуватись при реалізації алгоритму.

Оператори відділяються один від одного символом «крапка з комою»(;). Крапка з комою не являється частиною оператора, це розділювач операторів. Тому після останнього оператора програми та після останнього оператора в поставному операторі, тобто перед ключовим словом end, крапку з комою ставити не обов’язково.

Інують такі оператори: 1)Прості оператори(не можуть в собі містити інших операторів): присвоювання, процедури переходу.

2)Структурні оператори: складений оператор, умовний, процедури, приєднання.

<розділ операторів>::=begin <опис операторів>; end.

**Н-д:**  begin

Write(‘Hi KV’);

End.

**До структурних операторів належать:**

**1.Складений оператор.**

<Складений оператор>::=begin<оператор>{;<оператор>}end.

Структурні оператори включають в себе цілу групу операторів в єдине ціле, після чуго вони можуть раухватись одним оператором.Складений оператор складається з послідовності обєднуючих операторів, які розташовуються між ключовими словами begin та end.

**2.Умовний оператор if.**

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

Повна форма: if then else.

Неповна форма: if then.

Н-д: if a>b then if b>c then k:=1

Else k:=0;

Компілятор діє за правило: останнє else діє до останнього then.

Н-д: if a>b then if b>c then k:=1;

Else k:=0;

Всередині будь-якого структурного оператора ; стояти не можу, бо все що стоїть після ; вважається іншим оператором.

Перед ключовим словом else, ; не ставиться.

Найголовнішою перевагою умовного оператора є вкладеність

**3.оператор вибору case:**

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

<вираз-селектор>::=<вираз> - порядкового типу(до 65535 знач.), не може бути рядкового та дійсного типів.

<опис варіанту>::=<мітка>:<оператор>.

1.Обчислюєтсья значення виразу-селектора і запамятовуєтсья.

2.Серед міток варіанту відшукується мітка, що співпадає із значенням виразу – селектору і виконується оператор, помічений цією міткою, після чого управління передається на наступний за case оператор. Якщо значення виразу – селектора не співпало з жодною міткою варіанту, то управління передається або на наступний за case оператор, або на оператор після слова else.

**Білет№9 оператори повтору – while, repeat, for). Процедури, що змінюють хід роботи циклічного оператора.**

**Оператори:**

Оператори повтору:

<Оператор циклу>::=<цикл з передумовою>|<цикл з післяумовою>|<цикл з лічильником>

1.Цикл з передумовою

<цикл з передумовою>::=while <умовний вираз> do.

Порядок виконання циклу з передумовою:

1)Обчислюєтсья значення логічного виразу.

2)Якщо значення логічного виразу =Т, то виконується тіло циклу.

3)Доходячи до ; виконується пункт 1.

Зауваження: цикл з передумовою може ні разу не виконатись.

Приклад.

2.Цикл з постумовою:

<цикл з післяумовою>::=repeat<тіло циклу>until<логічний вираз>

Порядок виконання:

1)Виконується тіло циклу;

2)Обчислюється значення ум.виразу.

3)Якщо значення =false, то переходимо до пункту 1

Зауваження:1)Виконуєтся завжди як мінімум 1 раз. 2)В тілі циклу повинно бути те, що змінює значення умовного виразу на true.

Приклад.

3.Цикл з лічильником.

<>::=for<>to<>do<>;

<> <= <>

<>::=for<>downto<>do<>;

<> >= <>

<>::=<>

<>::=<>

<>::=<>

Процедури, що змінюють хід роботи циклічного оператора:  
break – вихід з циклу;

Continue – виконує перехід на нову ітерацію того циклу в тілі якого знаходиться;

Exit – виконує вихід з поточного блоку;

10**. Масиви. Описання масивів. Операції над масивами**

Масив - це упорядкований набір фіксованої кількості однотипних елементів. Кожен елемент масиву позначається ім'ям масиву з індексом. Елементи масиву впорядкувані за значеннями індексу. Масиви в Паскалі описюються наступним чином:

<імя типу> = array [<список. індекс. типів>] of <тип>;

Сумарна довжина будь-якого з типів не повинна перевищувати 65 520 байт

В якості індексу може бути вираз, окремим випадком якого являеться константа або змінна.

Елементи масиву можуть стояти як в лівій частині оператора привласнення, так і у виразах. Над елементами масиву можна робити ті ж операції, які допустимі для даних його його базового типу. Над масивами не визначені операції відносини, тому не можна записати А > В

Опис одномірних масивів:

var A: array [1..5] of integer

Опис двомірних масивів:

var A: array [1..5,1..5] of integer

12.**Записи. Фіксовані записи, їх описання в програмі. Операції над записами.**

Запис – це скінченна сукупність логічно повязаних різнотипових елементів. Дані одного поля різних записів мають один і той же тип, а різних полів можуть мати різні типи. Загальний вид опису типу <запис>::= record <список полів> end.

Приклад фіксовано запису:

Type strin = record  
 n:byte;  
 fio: array [1..20] of char;  
 mark: array [1..4] of {2,3,4,5};  
 end;  
Var student: strin;

Звернення до елементу записи виконується за допомогою складного імені: <імя запису>.<імя елементу>. Елементи запису використовуються як і звичайні змінні. Над елементами запису можна використовувати дії, які допустимі для даних його типу. Наприклад, якщо тип елементу запису – цілий, то виконуються всі операції допустимі для цілих даних.

11. **Рядки. Різні способи подання рядків. Поняття довжини рядка. Операції над рядками.**

Дані типу string. Вони використовуються для зберігання послідовностей символів. На відміну від масиву, кількість символів у рядку-змінній може змінюватися від 0 до N, де N – максимальна кількість символів у рядку. Значення N визначається оголошенням типу string[n] і може бути константою порядкового типу, але не більше 255. Можна не вказувати N, але в цьому випадку довжина рядка приймається максимально можливою, а саме N=255. Найперший байт в рядку має індекс 0 і містить поточну довжину рядка. Можна змінювати довжину рядка: string[0]:=5;

Як дізнатися довжину рядка: ord(s[0]); length(s);

З рядками можна виконувати операцію «+» - об'єднання (якщо довжина обєднаного рядка буду більша за N то зайві символи відкидаються).

Процедури і фукції:

Сoncat(s1,s2,..,sn) – функція типу string; повертає рядок, яка предаставляє собою обєднання рядків s1,s2,..,sn.

Copy(s,index,count) – функція типу string; копіює з рядка s count символів, починаючи з index позиції.

Delete(s,index,count) – процедура; видаляє count символів з рядка s, починаючи з символа на позіції index.

Insert (k,s,index) – процедура; вставляє підрядок k у рядок s, починаючи з символа на позіції index.

Length(s) – функція типу integer; повертаэ довжину рядка s;

Pos(k,s) – функція типу integer; шукаю у рядку s перше входження підрядка k і повертає номер позіції з якої вона починається; якщо підрядок не знайднено, то повертається 0.

Str (x, s) – процедура; перетворює числове значення x в рядок s, при цьому для x може бути заданий формат, як в процедурах виведення write і writeln. наприклад: x: = 123; s: = str (x: 6, s); Результат: s = '123'.

Val (s, x, k) – процедура; перетворить рядок s в значення числової змінної x, при цьому рядок s повинна містити символьне подання числа. У разі успішного перетворення зміна k дорівнює нулю. Якщо ж виявлена ​​помилка, то k буде містити номер позиції першого помилкового символу, а значення x не визначено.

13. **Записи. Варіантні записи, їх описання в програмі. Приклади.**

Запис – це скінченна сукупність логічно повязаних різнотипових елементів. Дані одного поля різних записів мають один і той же тип, а різних полів можуть мати різні типи. Загальний вид опису типу <запис>::= record <список полів> end.

Варіантні записи, в яких також, як і у фіксованих, описуються поля для всіх можливих випадків, однак альтернативні групи полів розмежовуються більш наочно і пам'ять виділяється тільки під конкретне, необхідний у даний момент варіант.

Загальний вигляд:

type v = record

a: type1

b: type2

case n: typen of

c1: type11;

c2: type21;

end;

var z: v;

Оператор приєднання дозволяє спростити звернення до поля запису. Ім'я запису виноситься в заголовок оператора приєднання, а в блоці використовуються тільки імена елементів запису. Загальна форма оператора присвоювання:   
with <імя запису> do <оператор>; Де with, do – зарезервовані слова; <імя запису> - імя зміної типу запис; <оператор> - довільний оператор Паскаля;

Приклад:

with student do   
 begin   
 n:=2;  
 s:=(mark[1]+mark[2]+mark[3]+mark[4])/4;  
 end;

**16. Множини. Описання множин. Операції над множинами, їх приорітет. Приклади.**

Поняття множини у програмуванні аналогічне математичному.Відмінність в тому, що в програмуванні множини мають обмежений розмір і елементи множини можуть бути обмежених типів.

Множина – це структурований тип даних, який представляє собою набір взаємнозвязаних за якимись ознаками елементів, який можна розглядати, як єдине ціле. Базовий тип – це той тип, до якого відносяться елементи множини.

Множина не містить однакових елементів, порядок слідування елементів не має значення.

<множина>::=set of <базовий тип>

<базовий тип>::=<порядковий тип>

Операції над множинами:

Порівняння(<=,<>..)

Перетину(\*)

Обєднання(+)

Різниця(-)

in

Пріоритет операцій:

1)\*

2)+-

3)=,<>,>=,<=,in

Приклади:

1. =

A[1,2,3,4]; B[2,1,4,3] – True

A[‘a’,’b’,’n’];B[‘a’,’n’] – False

1. <>

A[1,2,3];B[1,2,3,4] – True

B[‘c’,’t’];B[‘t’..’c’] – true

1. >=,<=

A[1,2,3,4];B[2,3,4] – true

A[‘a’..’z’];B[‘b’,’c’] – true

1. \*

A[1,2,3];B[2,3,] – [2,3]

1. +

A[1,2,3];B[4] – [1,2,3,4]

1. –

A[1,2,3];B[1,2,3] – [ ]

1. In

A[2];if A in [1,2,3] then .. True

Множину не можна вивести

1. **Рекурсія. Випереджаюче описання. Приклади.**

Рекурсія – такий спосіб організації обчислення процесу при якому підпрограму звертається сама до себе.

**1, if n=0**

**n! =**

**n(n-1)!, if n>0**

uses crt;

procedure PrintA;

begin

write(‘Рекурсія’);

PrintA;

end;

begin

PrintA;

end.

При написанні рекурсивної підпрограми особливу увагу треба приділяти виходу з підпрограми в потрібний момент.

Uses crt;

Function FAKT(i:integer):integer;

Begin

if i<0 then write(‘Error’)

else if i=0 then FAKT:=1

else FAKt:=Fakt(i-1)\*i;

end;

begin

writeln(‘Кількість поєднань з 2 по 1 =’,FAKT(2)/(FAKT(1)\*FAKT(2-1));

end.

Можна показати, що будь-яка задача, яка розв’язується рекрсивно може бути розв’язана також ітераційно, але ці 2 підходи ідуть до розв’язання з протилежних боків.

Рекурсивно: від складного до простого.

Ітераційно: від простого до складного.

Ітерація

Function FIB(n:integer):longint;

var a1,a2:longint;

i:integer;Result:integer;

begin

A1:=0;A2:=0;

If n=0 then Result:=0;

Else Result:=1;

For i:=2 to n do

Begin

Result:=A1+A2;

A1:=A2;

A2:=Result;

End;

FIB:=Result;

End;

Рекурсія

Function Fibr(n:integer):longint;

Begin

If n<2 then Result:=n;

Else Result:=Fibr(n-1)+Fibr(n-2);

Fibr:=Result;

End;

Кожен рекурсивний виклик вимагає додаткового часу. При кожному виклику для локальних змінних і для параметрів значень виділяється нова пам’ять.

Uses crt;

Const k=1;

Function SUMA(res,i,n:byte):byte;

Begin

Res:=res+(i+k);

If i=n then SUMA:=res;

Else SUMA:=Suma(res,i+1,n);

End;

Begin

Writeln(Suma(0,1,3));

End.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Рівень рекурсії | Res | i | n | SUMA |
| Виклик |  |  |  | SUMA(0,1,3) |
| 1 | 0+1+1 | 1 | 3 | SUMA(2,2,3) |
| 2 | 2+2+1 | 2 | 3 | SUMA(5,3,3); |
| 3 | 5+3+1 | 3 | 3 | SUMA:=9 |

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

Існує 2 поняття файлів: зовнішній(логічний) і внутрішній(фізичний).

Фізичний – це іменована частина файлу.

Внутрішній – той, що описаний в програмі. Він представляє собою структуру даних.

Файловий тип – це єдиний тип який може пов’язати програму з зовнішніми прситроями.

Класифікація файлів:

**За способами доступу За типом**

Файли прямого доступу Типізовані

Файли послідовного доступу Нетипізовані

Текстові

Описання для типізованих файлів

<ім’я типа>::= file of <тип елементів>

Нетипізованих файлів

<ім’я типа>::= file

Текстовий файл

<ім’я типа>::= text

Type

MORZE=(tochka, tyre);

Telegram=file of MORZE;

Var

F: Telegram;

Letter: file of char;

Відмінності

Масив Файл

1. Пам’ять

Оперативна Постійна

2. Розмір

Постійний і визначений

при описанні Змінюється, перший - О останній – eof (ctrl-Z)

1. Assign(F, ‘1.dat’) – встановлює зв’язок між фізичним та логічним файлом. Виконується лише для закритих файлів.
2. Rewrite(F) – готує файл дя запису і створює вікно на першу позицію.
3. Reset(F) – відкриває існуюий файл для читання

Текст – для читання

Нетекст – для читання/запису

1. Close(F)
2. eof(F) – повертає значення true
3. Rename(F)

Після assign, але до Reset або після Close

1. Erase(F) – знищує фізичного файла, для закритих файлів
2. Write(F,x)
3. FilePos(F) – повертає номер поточної позиції файла для файлів прямого доступу.
4. FilseSize(F) – кількість елементів.
5. Seek(F,i) – встановлює вікно файла на задану позицію.
6. Tuncate(F) – для відкритого файлу

Підрізає файл до поточної позиції

19.**Типізовані файли прямого доступу. Типізовані файлі послідовного доступу. Приклади.**

За формою доступу до компонентів розрізняють файли прямого доступу і файли

послідовного доступу. Для файлів послідовного доступу характерним є те, що

його компоненти можуть опрацьовуватись лише в природному порядку (тобто

порядку їх слідування). Покажчик файла при виконанні операції читання або

запису переміщується до наступної компоненти.

Для файлів прямого доступу існує можливість доступу до довільного компонента.

Оскільки всі компоненти файла мають однакову довжину, положення будь-якої

компоненти легко можна обчислити.

**Приклади:**

**1)прямого доступу**

1. **program** Project1;
2. **сonst**
3. Fn = 'file.dat';
4. **var**
5. F : **file** **of** **Integer**;
6. i, Num, NumEnd, Size : **Integer**;
7. **begin**
8. Writeln('Файл: ', Fn);
9. Assign(F, Fn);
10. Rewrite(F);
11. Writeln('Содержимое файла до обработки:');
12. **for** i := 0 **to** 9 **do** **begin**
13. Write(F, i);
14. **if** i > 0 **then** Write(', ');
15. Write(i);
16. **end**;
17. Writeln;
18. Seek(F, 0);
19. **if** **not** Eof(F) **then** **begin**
20. Read(F, Num);
21. Size := FileSize(F);
22. Seek(F, Size - 1);
23. Read(F, NumEnd);
24. Seek(F, Size - 1);
25. Write(F, Num);
26. Seek(F, 0);
27. Write(F, NumEnd);
28. **end**;
29. Writeln('Содержимое файла после обработки:');
30. Seek(F, 0);
31. i := 0;*.*
32. **while** **not** Eof(F) **do** **begin**
33. Read(F, Num);
34. **if** i > 0 **then** Write(', ');
35. Write(Num);
36. **if** i = 0 **then** i := 1;
37. **end**;
38. Close(F);
39. Readln;
40. **end**.
41. **2)послідовного доступу:**
42. **uses** crt;
43. **var** f,c:**file** **of** **word**;
44. **procedure** Make\_File;
45. **var** x,i,j,b:**word**;
46. **begin**
47. clrscr;
48. readln(b);
49. assign(f,'n1\_a.dat');
50. rewrite(f);
51. assign(c,'n1\_b');
52. rewrite(c);j:=1;
53. randomize;
54. **for** i:=1 **to** b **do** **begin** x:=random(9)+1;
55. write(f,x);
56. j:=j\*x;write(c,j);
57. **end**;close(f);
58. close(c);
59. **end**;
60. **procedure** view;
61. **var** i,j:**word**;**begin** reset(f);
62. reset(c);
63. **for** i:=1 **to** filesize(f) **do** **begin** read(f,j);
64. writeln('f=',j);
65. read(c,j);
66. writeln('c=',j);
67. **end**;
68. readln;
69. **end**;
70. **begin** make\_file;
71. view;

**end**.

20.**Текстові файли. Нетипізовані файли. Приклади**.

**Текстові файли - файли на диску, що складаються з символів ASCII. Для розділення рядків використовуються символи «кінець рядка». Текстові файли є файлами з послідовним доступом. У будь-який момент часу доступна тільки один запис файлу. Інші записи стають доступними лише в результаті послідовного просування по файлу. Текстові файли внутрішньо розділені на рядки, довжини яких різні. Для розділення рядків використовується спеціальний тег кінця рядка. Оголошуються текстові файли типу змінної text. Обробляти їх можна тільки послідовно і з допомогою процедур і функцій:**

**Readln (f , st )- читання рядка st з файлу f і перехід на початок наступного ;**

**Writeln (f, st )- запис рядка st у файл f і маркера кінця рядка ;**

**Append (f ) - процедура, що відкриває файл f для додавання рядків в кінець файлу;**

**Eoln (st )- логічна функція, результат виконання якої дорівнює TRUE, якщо досягнуто маркер кінця рядка st.**

**Приклад 1. Створити текстовий файл, у який записати 3 пропозиції. Прочитати цей файл, вивести його вміст на екран. Визначити довжину кожного пропозиції.**

**Program File\_text;**

**var**

**f1 : text;**

**st : string;**

**n: byte;**

**begin**

**assign (f1, "file1.txt"); {зв'язати з файлом file1.txt файлову змінну f1 }**

**rewrite (f1); { створити новий файл з ім'ям file1.txt }**

**writeln ( f1, "Дуже корисно вивчати"); { записати пропозиції у файл}**

**writeln ( f1, " всім студентам ");**

**writeln (f1, мовою Pascal ");**

**close (f1); { закрити файл для запису }**

**reset (f1); { відкрити файл для читання }**

**while not eof (f1) do { поки не кінець файлу f1}**

**begin**

**readln (f1, st); {читаємо рядок з файлу f1 }**

**writeln(st); { виводимо на екран }**

**n:= length (st); {визначаємо довжину рядка }**

**writeln (" довжина =",n);**

**end;**

**close (f1); { закрити файл для читання} end .**

**Нетипізовані файли**- це послідовність компонент довільного типу.

Відкриття нетипізовані файлу:

Reset(f, BufSize) Rewrite(f, BufSize)

Параметр BufSize задає число байтів, що записуються в файл за одне звертання або зчитуються з нього. Мінімальне значення BufSize - 1 байт, максимальне - 64 Кбайт. Якщо пропустити цей параметр, то він отримає типове значення рівне 128 байтам.

Читання даних з нетипізований файлу:

BlockRead(f, X, Count, QuantBlock);

Ця процедура здійснює за одне звертання зчитування в змінну X такої кількості блоків, яка задана параметром Count, при цьому довжина блоку дорівнює довжині буфера. Значення Count не може бути менше 1. За одне звернення не можна прочитати більше, ніж 64 Кбайт.

Необов'язковий параметр QuantBlock повертає число блоків, прочитаних поточною операцією BlockRead. У разі успішного завершення операції читання QuantBlock = Count, у разі аварійної ситуації параметр QuantBlock буде містити число вдало прочитаних блоків. Звідси випливає, що за допомогою параметра QuantBlock можна контролювати правильність виконання операції читання.

Запис даних у нетипізований файл:

BlockWrite(f, X, Count, QuantBlock);

Ця процедура здійснює за одне звертання запис зі змінної X кількості блоків, яка задана параметром Count, при цьому довжина блоку дорівнює довжині буфера.

Необов'язковий параметр QuantBlock повертає число блоків, записаних успішно поточною операцією BlockWrite.

Для нетипізованих файлів можна використовувати процедури Seek, FIlePos і FileSize, аналогічно до відповідних процедур для типізованих файлів.

21. **Процедурні типи, їх використання.**

Type< ідентифiкатор>=procedure[ ( <формальні параметри>)]

<ідентифікатор>=function[ (<формальні параметри>)]:тип функции ;

Приклад

Type

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

Nonparam=procedure;

Var pp:Proc ; FF:fun; P1:nonparam;

Вкладені виклики

Type errorFunc=function(code:integer):Boolean;

FillFunc=function(index:integer,F:errorFunc):byte;

Змінним процедурного типу можна присвоювати значення іншої процедурної змінної або ідентифікатора процедури чи функції.

Приклад

{$F+}

Function Multiply(x,y:integer):real;

Begin multiply=x+y; end;

{$F-}

Begin

F:Multiply;

Writeln(‘ Множення 30 та 24=‘,);

F(24,30);

End;

Для виконання такого присвоєння треба дотримуватися таких вимог:

1)однакова кількість параметрів;

2)параметри повинні мати однаковий тип;

3)Процедури або функції не повинні біти стандартними або визначеними в якомусь модулі;

4)Така процедура чи функція не повинна бути вкладеною;

5)Не повинна бути типу inline. Та процедурою переривання Interrupt.

Type myfunc=function(x,n:longint):longint:

{$F+}

Function Step(x,n:integer):longint;

Begin

Step:=n\*ln(x);

End;

{$F-}

Const step :myfunc=step;

Фактично змінна процедурного типу представляє собою адресу пам’яті де розмістилася процедура або функція і таким чином є вказівником особливого виду. Для отримання адреси ,де міститься виконуваний код процедури або функції треба виконати операцію взяття адреси . Директиви nil і far вмикають різні моделі виклику процедур для генерації вихідного коду. 22. **Вказівники. Константа nil. Породження та знищення динамічних об’єктів. Дії над вказівниками. Стандартні процедури та функції для роботи з вказівниками.**

Вказівник - це змінна, яка містить адресу іншої змінної (байта пам'яті). У ТР є два види вказівників: вказівник на об'єкт деякого типу (типізований) і вказівник, не пов'язаний з типом (pointer). Вказівник – це спеціальний тип. Значення цього типує адреса програмного обєкта за якою відбувається доступ до цього обєкту. Описується вказівниквий тип так: type<ідентифікатор типу> = ^<базовий тип>

Для змінних типу вказівник введено стандартне значення NIL, яке означає, що покажчик не посилається ні до якого об'єкту. Константа NIL використовується для будь-яких покажчиків.

Процедура new (A) виділяє область пам'яті відповідно того типу, який описаний для вказівника А і записує адресу виділеної пам'яті у вказівник.

Процедура dispose(A) вивільняє область памяти, на яку вказував вказівник А, після чого ця область памяті стає доступною для розподілення під інші динамічні змінні.

Для вказівників допустимі операції :

1. Порівняння і присвоювання. Вказівником можна присвоїти вміст іншого вказівника того ж самого типу або константу NIL - порожній, або адреса об'єкту за допомогою функції ADDR або оператора @.

P1 := PP;   
P2 := NIL;  
P3 := Addr(X);  
P4 := @X;

Функції:

ADDR (X) - результат POINTER, в якому міститься адреса аргументу. (X-ім'я будь-якої змінної, процедури або функції).

MAXAVAIL: LONGINT - повертає розмір найбільшого безперервної ділянки купи.

MEMXAVAIL: LONGINT - повертає розмір загального вільного простору купи.

 Процедури:

DISPOSE (TP: POINTER) - знищує динамічну змінну і повертає в купу фрагмент динамічної пам'яті, який був зарезервований вказівником.

NEW (TP: POINTER) - резервує фрагмент купи для розміщення змінної.

GETMEM (P: POINTER; ZIZE: WORD)-виділяє з купи блок заданого розміру та адресу його початку привласнює вказівником.

FREEMEM (P: POINTER; ZIZE: WORD) - звільняє блок заданого розміру ..

MARK (P: POINTER) - запам'ятовує поточну вершину купи (адреса початку вільної ділянки).

RELEASE (P: POINTER) - звільняє ділянку купи від адреси з P до кінця.

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

Delphi и Pascal нельзя назвать двумя принципиально разными языками программирования, фактически, Delphi - это тот же паскаль, просто Delphi - язык(среда разработки) более высокого уровня, который был разработан в связи с потребностями программистов в более удобной и функциональной среде для разработки сложных программных продуктов. Одним из главных и приятных преимуществ Delphi является его многопоточность, что позволяет программисту создавать сложные проекты, которые будут взаимодействовать с сетью и не создать пользователю дискомфорт. Так же Delphi можно назвать и событийно-ориентированным языком, поскольку там реализована обработка событий, которая работает по схеме MVC( Modal-View-Controller). Для примера можно показать обработку событий взаимодействия с мышью( более 10 видов событий таких как зажатие, двойной клик,…). И поскольку это язык более высокого уровня, в нем уже изначально заложены и реализованы методы классов( их можно переопределять), которые намного упрощают жизнь программисту и позволяют сделать код короче и визуально красивее, а проект сложнее и функциональней. Это только некоторые преимущества среды разработки Delphi.

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

Об’єктно-орієнтоване програмування – це методологія програмування, яка основана на представленні програми у вигляді сукупності об’єктів, кожний з яких являється реалізацією певного класу, а класи створюють ієрархію на принципах наслідування.

ООП виникло як відповідь на зростаючу складність та об’єм програм. Призначенням було дисциплінувати процес створення великих програмних комплексів і тим самим знизити їх вартість та складність.

Переваги ООП:

1)Економія часу та ресурсів за рахунок використання вже написаних компонентів

2)Значно менша кількість помилок в компонентах програми, що вже не раз тестувалися при використанні у інших проектах

3)Можливість вести незалежну розробку програми

4)Полегшення аналізу та проектування програм

Змінні об’єктних типів називаються екземплярами об’єктів, так само як записи представляють собою комбіновані структури даних.

**TPosition = Object**

**Column : integer;**

**Row : integer;**

**End;**

Обєкт характеризується як сукупність своїх властивостей так і сукупністю допустимих для даного об’єкта дій.

І структурна, і ОО-методологія мають на меті побудову ієрархічного дерева взаємозв’язків між об’єктами. Але якщо структурна ієрархія будується по принципу розділення цілого на складові частини, то при створенні ОО-ієрархії обов’язково відображається наслідування властивостей батьківських типів об’єктів дочірніми типами об’єктів.

1)Наслідування – це властивість системи, що дозволяє описати новий клас на основі вже існуючих, що повністю або частково запозичує в них функціонал. Клас від якого здійснюється наслідування називається базовим, батьківським або суперкласом. Новий клас – нащадком, дочірнім або похідним класом.

2)Інкапсуляція – це властивість системи, що дозволяє обєднати дані та методи, які працюють з ними в класі, та приховати деталі реалізації від користувача.

3)Поліморфізм – це властивість системи використовувати об’єкти з однаковим інтерфейсом без інформації про тип та внутрішню структуру об’єкта.(Властивість різних об’єктів виконувати одну й ту саму дію по-своєму).

Область дії полів та методів:

Поля та методи поділяють одну й ту саму область дії, не зважаючи на те, що вони можуть бути описані в різних місцях тексту програми.

Якщо який-небудь екземпляр типу ObjectVar викликає метод Method, що належить йому, то виконується неявний оператор

With ObjectVar do Method

який зв’язує об’єкт і його методи в одній області дії.

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

Статичні методи – такі методи, адреса яких зв’язується з екземплярами об’єктів статично, ще під час компіляції і не змінюється до завершення роботи. Коли викликається статичний метод, для реалізації методу використовуються дані часу компіляції.

Раннє зв’язування – процес статичного зв’язування методів з реалізації об’єктів.

Пізднє зв’язування – процес динамічного зв’язування методів з реалізаціями об’єктів.

Віртуальні методи використовують пізнє зв’язування, статичні – раннє.

Синтаксично віртуальні методи відрізняються від динамічних наявністю директиви virtual. Віртуальні методи оптимізовані по швидкодії, а динамічні – по об’єму коду.

Якщо оголошується віртуальний метод в батьківському типі, то це накладає обмеження на його дочірні типи:

* Усі однойменні методи дочірніх типів також повинні бути віртуальними
* Заголовки одного методу повинні бути ідентичними, включно з числом параметрів і їх типів
* Кожен об’єкт з віртуальними методами повинен мати конструктор

Конструктор – це спеціальний тип процедури, яка виконує початкові установки для роботи механізму спеціальних методів.

Кожен окремий екземпляр об’єкту має бути ініційований окремим викликом конструктора.

Деструктор – спеціальний тип процедури, «збірник сміття». Призначений для видалення динамічно розміщенних об’єктів та звільнення памяті, що вони займають.

Деструктори можуть бути як статичними, так і віртуальними. Їх можна наслідувати.

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

Динамічними називають такі програмні об’єкти, які виникають в процесі виконання програми.

Аналогічно динамічним змінним будь-яких інших типів пам'ять під динамічні об’єкти виділяється процедурами New і Dispose:

**Var**

**PWKg, PBKg : ^TKing;**

При використанні динамічних об’єктів зручно працювати з розширеним синтаксисом процедур New і Dispose.

Перше розширення дозволяє викликати New як функцію:

**Type**

**PTKing = ^TKing;**

**Var**

**PWKg, PBKg : PRKing;**

**Begin**

**PWKg := New(PTKing);**

**PBKg := New(PTKing);**

**…**

**End.**

Друге розширення дозволяє використовувати другий параметр, який вказує конструктору імя конструктора з фактичними параметрами, що ініціює динамічний об’єкт відразу ж після виділення для нього область памяті.

**New (PWKg, Init (‘e’, 1, WhiteColor ));**

**PWKg^.Display;**

Розширення Dispose аналогічно другому розширенню процедури New. Відмінність полягає в тому, що другий параметр є імя не конструктора, а деструктора.

**Dispose(PWKg, Done);**

Помилки під час роботи з динамічними об’єктами

1. Під час виклику процедури New с двома параметрами імя конструктора не може бути складеним, адже під звернення динамічний об’єкт ще не створено.

Наприклад оператор

**New(Pline, PLine^.Init);** видасть повідомлення про помилку

2. Якщо не вистачає памяті для розміщення екземпляра динамічного об’єкта, то виклик конструктора генерує фатальну помилку з кодом 203.

HeapFunc – функція, що дозволяє обробляти помилки динамічно розподіленої області памяті.

**function HeapFunc(Size: word): integer; far;**

Ця фунція реалізується за допомогою присвоєння її адреси змінній HeapError:

**HeapError := @HeapFunc;**

Якщо параметр **Size** містить розмір блоку, для якого не знайшлося області памяті відповідного розміру, то функція обробки памяті спробує звільнити блок, розмір якого не менше.

**Директива private**

Директива private в рамках описання об’єкта виконує таку ж роль, як і розділ реалізації implementation в модулі. Тобто призначена для приховування деяких деталей об’єкта від користувача. Приватні поля і методи доступні тільки всередині того модуля, в якому описаний об’єкт.

**Type**

**ObjectType = object**

**Загальнодоступні поля;**

**Загальнодоступні методи;**

**private**

**Приватні поля;**

**Приватні методи;**

**End;**

**Директива public**

Директива publicробитьполя та методи загальнодоступними.

За замовчуванням усі поля та методи є загальнодоступними. Директива public дозволяє змінити порядок описання полів та методів.

**Type**

**ObjectType = object**

**private**

**Приватні поля;**

**Приватні методи;**

**public**

**Загальнодоступні поля;**

**Загальнодоступні методи;**

**End;**

**Директива protected**

Робить поля і методи доступними тільки для цього класу та нащадків цього класу, проте приховує від користувачів.26. **Концепція RAD. Система розробки програм Delphi та її IDE.**

RAD (*rapid application development*) – це концепція створення засобів розробки програмних продуктів, що виділяє особливу увагу на швидкість і зручність програмування, створенню технологічного процесу, що дозволяє програмісту максимально щвидко створювати програми.

Ця концепція виникла в кінці 20 століття, як відповідь на незграбні методи програмування 70-80 років, передбачали настільки повільний процес створення програми, що навіть вимоги до програми встигали змінюватися ще до завершення її розробки. RAD передбачає, що розробка ПЗ здійснюється невеликою командою протягом 3-4 місяців з використання візуального програмування. Однією з особливостей є залучення замовника вже на ранніх стадіях розробки.

Основні принципи:



* Створення *прототипу* для уточнення вимог замовника
* Циклічність розробки: кожна нова версія продукту базується на оцінці результата роботи попередньої версії програми замовником
* Мінімізація часу розробки версії за рахунок переносу готових модулів і додавання функціоналу в нову версію
* Тісна співпраця команди розробників
* Інструментарій повинен бути націлений на мінімізацію часу розробки

Delphi – це інтегрована середа швидкої розробки програмного забезпечення для Microsoft Windows, Mac OS, iOS и Android на мові Delphi(Object Pascal). Створена фірмою Borland, зараз належить Embarcadero Technologies.

Завдяки унікальному поєднанню простоти мови та генерації машинного коду дозволяє безпосередньо та на досить низькому рівні взаємодіяти з операційною системою та бібліотеками.

Середовище Delphi

* Головне меню
* Панель інструментів
* Палітра компонентів
* Інспектор об’єктів
* Вікно форми
* Редактор коду

**27.** **Візуальна технологія розробки програм.**

Візуальне програмування - програмування, що передбачає створення

додатків за допомогою наглядних засобів. При цьому програміст показує, що

необхідно отримати в результаті, а текст програми генерується автоматично

за допомогою візуального прототипу.

Застосування візуальних об’єктно-орієнтованих засобів розробки

дозволило різко скоротити час розробки завдяки використанню стандартних

об’єктів API, раніше розроблених об’єктів і ін. Недоліком стало те, що

стало неможливим уявити, що відбувається при виконанні програми.

Основна проблема програмування сьогодні – переносиміть

програмного забезпечення між комп’ютерами, які працюють під різними

операційними системами, мобільними устоями, ноутбуками та іншими

пристроями. Рішення цієї проблеми пов’язано з появою платформи .NET

(dotNet, що вимовляється як «Дот-Нет») та мов програмування для неї – C#

(C sharp, говорять «сі шарп»), Delphi .NET й ін.

Недоліком є те, що виконання програми стає некерованим процесом і

залежить від ефективності платформи .NET.

Існує багато систем візуального програмування, серед яких найбільш

вдалими є наступні проекти:

Microsoft Visual Basic

Microsoft Visual C++

Microsoft Visual C#

Borland Delphi

Середовища для програмування JAVA, такі як IDEA, Sun One Studio,

Borland Jbuilder, NetBeans та інші.

Головні складові частини середовища програмування *Delphi*:

* Дизайнер форм (*Form* *Designer*)
* Вікно текстового редактора тексту (*Editor* *Window*)
* Палітра компонент (*Component* *Palette*)

Інспектор об'єктів (*Object* *Inspector*)

**28.** **Подіє-орієнтоване програмування.**

Событийно-ориентированное программирование (event-driven programming) — это парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

Событийно-ориентированное программирование можно также определить как способ построения компьютерной программы, при котором в коде (как правило, в головной функции программы) явным образом выделяется главный цикл приложения, тело которого состоит из двух частей: выборки события и обработки события.

Событийно-ориентированное программирование, как правило, применяется в трех случаях:

1)при построении пользовательских интерфейсов (в том числе графических);

2)при создании серверных приложений в случае, если по тем или иным причинам нежелательно порождение обслуживающих процессов;

3)при программировании игр, в которых осуществляется управление множеством объектов.

**Событийно-ориентированное программирование.**

С активным распространением системы *Windows*и появлением визуальных *RAD-*сред широкую популярность приобрел событийный подход к созданию программ – *событийно-ориентированное программирование.*

Идеология системы *Windows*основана на событиях. Щелкнул человек на кнопке, выбрал пункт меню, нажал на клавишу или кнопку мыши — в *Windows*генериру­ется подходящее *сообщение,*которое отсылается окну соответствующей программы.

Структура программы, созданной с помощью событийного программирования, следующая. Главная часть представляет собой один бесконечный цикл, который опрашивает *Windows,*следя за тем, не появилось ли новое сообщение. При его обна­ружении вызывается подпрограмма, ответственная за *обработку*соответствующего события (обрабатываются не все события, их сотни, а только нужные), и подобный цикл опроса продолжается, пока не будет получено сообщение «Завершить работу».

События могут быть *пользовательскими,*возникшими в результате действий поль­зователя, *системными,*возникающими в операционной системе (например, сообще­ния от таймера), и*программными,*генерируемыми самой программой (например, обнаружена ошибка и ее надо обработать).

Событийное программирование является развитием идей нисходящего проекти­рования, когда постепенно определяются и детализируются реакции программы на различные события.

**Событийно-ориентированное программирование**– это способ построения компьютерной программы, при котором в коде (как правило, в головной функции программы) явным образом выделяется **главный цикл приложения**, тело которого состоит из двух частей: **выборки события** и **обработки события**.

Событийно-ориентированное программирование, как правило, применяется в трех случаях:

при построении пользовательских интерфейсов (в том числе графических);

при создании серверных приложений в случае, если по тем или иным причинам нежелательно порождение обслуживающих процессов;

при программировании игр, в которых осуществляется управление множеством объектов.

Примеры реализаций:

Веб-серверы:

nginx

lighttpd

Прокси-серверы:

Squid

Применение в интерфейсах. СОП может быть как расширением структурного, так и объектно-ориентированного программирования (в дальнейшем ООП). Эпостасиями СОП являются cобытия и их приоритеты. События — специльные процедуры (для структурного програмирования) или методы (для ООП). События можно разделить на обычные, выполнение которых начинается по их прямому вызову (пользователя или программы) и события-демоны, находящихся в фоновом режиме и выпняющихся при совершении события, например, выполнения условий. При наступления ситуации, когда событие должно начать работу оно уступает событиям, имеющий больший приоритет. Приоритет может быть выражден от динамического до статического, крайним случаем которого является порядок следования событий.

Языки программирования. Разные языки программирования поддерживают СОП в разной степени. Наиболее полную поддержку событий обладают следующие языки (неполный список):

Perl (события и демоны DAEMON, и их приоритеты PRIO),

Java,

Delphi,

C# (события event).

Классы событий могут наследоваться (для исключений необходимо определить класс, как класс производный от системного: у С# системным является Exception, у С++ ).

Остальные языки, в большей их части, поддерживают события как обработку исключительных ситуаций.

29. **Мова програмування Delphi (Object Pascal).**

**Object Pascal** — [об'єктно-орієнтована](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%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) мова програмування, нащадок [Pascal](http://uk.wikipedia.org/wiki/Pascal), більш відома як основна [мова програмування](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%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) середовища [Delphi](http://uk.wikipedia.org/wiki/Embarcadero_Delphi). В 2001 році компанія Borland відмовляється від назви Object Pascal і називає мову програмування, що використовується в [Borland Delphi](http://uk.wikipedia.org/wiki/Borland_Delphi), [мовою програмування **Delphi**](http://uk.wikipedia.org/wiki/Delphi_(%D0%BC%D0%BE%D0%B2%D0%B0_%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)).

Delphi - це середовище розробки додатків for Windows.Delphi підтримує об'єктно-орієнтоване і подієвепрограмування.  
  
Delphi - об'єктно-орієнтована мова програмування зможливістю доступу до метаданих класів (тобто до описукласів і їх членів) в компільованому коді, що такожназивається інтроспективною. Його базовою мовою ємова Object Pascal. Він приніс в мову Паскаль багатонового.