**Житомирський військовий інститут**

**Національного авіаційного університету**

# Реферат

# на тему:

Структуровані типи даних: записи та множини. Використання процедур та функції мови Turbo Pascal. Поняття про записи. Прості записи

Житомир 2010

**План**

1. Ієрархічні записи
2. Записи з варіантними полями
3. Множини
4. Загальні відомості про процедури та функції
5. Процедури з параметрами
6. Процедури без параметрів
7. Особливості використання функцій
8. Методи звертань до процедур і функцій
9. **Поняття про записи. Прості записи**

Записи призначені для складання програм, які забезпечують дії з таблицями, відомостями, списками, каталогами і т.д., де містяться дані різних типів.

Розглянемо суть запису на прикладі 1:

Приклад 1:

Нехай маємо таблицю відомостей про типи зброї:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип зброї | Індекс зброї | Дальність дії | Висота дії |
| N | W | I | D | H |

Кожному рядку такої таблиці відповідає запис, який має своє власне ім’я (також, як і змінна), наприклад: TWOO. Запис складається із окремих часток відповідно стовпчикам таблиці – поля записів (у прикладі 1 5 полів).

Типи даних, які записують у полях, можуть бути різноманітними. Для прикладу 1:

№ - тип INTEGER**;**

тип зброї W – може бути обмеженим, який вводиться оператором, наприклад WOOR (або може використовувати тип STRING);

індекс I – також може бути обмежений цілочисловий (IND);

дальність D та висота H – типу REAL.

Таким чином, тип записів, як такий, не існує. Він містить різнотипові елементи.

Кожне поле має складене ім’я з двох компонентів, які відокремлюються крапкою:

**IZ.IP.**

Для прикладу 1:

**TWOO . N;**

**TWOO . W;**

**TWOO . I;**

**TWOO . D;**

**TWOO . H.**

Опис записів виконують двома способами:

1. у розділі типів та змінних;
2. у розділі змінних.

**!Особливість!** необхідно спочатку описати не стандартні типи даних ( наприклад **WOOR, IND**).

Порядок опису записів:

1. у розділі типів та змінних:

type

**<ім’я типу> =** RECORD

**<ім’я поля> : <тип поля>;**

**……………………………..;**

**<ім’я поля> :<тип поля>;**

end**;**

var **<ім’я типу>**

1. у розділі змінних:

var **<ім’я типу> :** RECORD

**<ім’я поля> : <тип поля>;**

**……………………………..;**

**<ім’я поля> : <тип поля>;**

Для прикладу 1 маємо:

1. type

**WOOR = (str, art, zrk, avia);**

**IND = (1..100);**

**TZW =** RECORD

**N :** integer**;**

**W : WOOR;**

**I : IND;**

**D, H :** real**;**

End**;**

Var

**YWOO : TZW;**

Або 2-й спосіб:

Type

**WOOR = (str, art, zrk, avia);**

**IND = (1..100);**

Var

**TWOO :** RECORD

**N :** integer**;**

**W : WOOR;**

**I : IND;**

**D, H :** real**;**

End**;**

Особливість:

а) над кожним складеним іменем можна виконувати будь-які операції, допустимі для даного типу, до якого відносяться дані поля;

б) на тип полів не накладаються ніякі обмеження. Таким чином для полів використовують як складні так і прості типи даних. В частості поля можуть бути масивами або записами.

Приклад 2:

Розробити програму для обчислення щільності (густини) потужності активної завади передавача завад за формулою

**P = (P\*G)/f**;

де **Р** – потужність передавача завад;

**G** – коефіцієнт підсилення;

**f** - ширина спектра завад.

Дані про передавач завад можна занести до таблиці:

|  |  |  |
| --- | --- | --- |
| **Р** | **G** | **f** |

Тобто представимо у вигляді трьох полів: **P, G, f**. Ім’я запису визначимо як **PMAP**. Тип даних полів як REAL.

Такий принцип використання імен значно збільшує об’єм тексту програми, тому в мові ТР передбачений спеціальний оператор приєднання WITH, який виносить наперед ім’я запису, а всередині оператора використовується тільки ім’я полів.

Формат оператора приєднання:

WITH **<ім’я запису>** DO **<оператор>.**

Для прикладу 2 змінюються оператори виведення та присвоюваня:

WITH **PMAP** DO

Begin

Read **(P, G, Df);**

**RO := P\*G/Df;**

End**;**

Внутрішнім оператором виступає складений оператор BEGIN**…**END**.**

Program **plotnost(input, output);**

Var

**RO :** real**;**

**P, G, Df :** real**;**

End**;**

Begin

WriteLn**(‘ввести таблицю P, G, Df’);**

Read **(PMAP . P, PMAP . G, PMAP . Df);**

**RO := PMAP . P \* PMAP . G/PMAP . Df;**

WriteLn **(‘щільність RO = ‘, RO);**

End**.**

Таким чином, для звертання до полів у програмі використовується складне ім’я:

**<ім’я запису> . <ім’я поля>**

**PMAP . P;**

**PMAP . G;**

**PMAP . Df.**

Таким чином при використанні оператора WITH збільшується швидкість використання програми, бо ім’я запису **PMAP** в пам’яті буде обчислене тільки один раз при обробці конструкції WITH **PMAP DO.**

Ще більшої ефективності можна досягнути, якщо запис є, наприклад, елементом масиву:

Var

**M : array[1..100]** ofchar**;**

Begin

With **M[ i ]** do

Якщо фрагмент програми містить операції над декількома записами, то в операторі приєднання можна указати список необхідних імен записів через кому:

WITH **V1, V2, V3** DO **S**

де **V1, V2, V3** – імена записи;

**S** – операнд.

Така форма еквівалентна :

WITH **V1** DO

WITH **V2** DO

WITH **V3** DO **S.**

Особливість:

1. якщо деякий ідентифікатор **F** є полем**,** він інтерпретується як **V2.F**, тобто ідентифікатор поля зв’язується з останнім записом у списку WITH.
2. всі ідентифікатори в межах оператора приєднання перевіряються на можливість їх інтерпретації як полів записів, заданих у заголовку.
3. **Ієрархічні записи**

Поля окремих записів також можуть бути записами. Тому складені імена можуть містити не дві, а більшу кількість компонент, тобто мати ієрархічний характер.

Наприклад:

**A . B . C;**

**A . B[ i ] . C** і т.д.

Розглянемо приклад : відомості про студентів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Група | Прізвище | Ім’я | По батькові | Рік народження |
| N | GR | F | I | O | GOD |

Присвоюємо ім’я запису W, який буде містити 6 полів. Поєднаємо поля F, I, O у запис з іменем FIO, який буде мати 3 поля і входити у запис W.

Опис такого запису можна представити так:

Type

**SPISOK =** string[50];

**GROUP = (GR1, GR2, GR3, GR4);**

**IND = (1980..1985);**

Var

**W :** RECORD **{опис запису вцілому}**

**N :** integer**;**

**GR : groupe;**

**FIO :** RECORD **{опис внутрішнього запису}**

**F : SPISOK;**

**I : SPISOK;**

**O : SPISOK;**

**{кінець внутрішнього запису}**

End**;**

**GOD : IND**

End**;{кінець запису}**

Тоді в розділі операторів можна поставити конструкцію:

WITH **W, FIO DO**

**N := 1;**

**GR := 3;**

**F := ‘ Кравченко’;**

**I := ‘Сергій’;**

**O := ‘Петрович’;**

**GOD := 1980;**

1. **Записи з варіантними полями**

Взагалі запис може складатися з двох частин: фіксованої і варіантної (тобто декілька варіантів структури цього типу). Варіантна частина містить декілька альтернатив, кожна з яких у круглих дужках задає список полів даного варіанта. Списку передує мітка, яка є конкретним значенням поля – критерій вибору варіантів. Перелік альтернатив починається з визначення цього поля.

Приклад 3: нехай **TIP** – ім’я змінної типу **Т**, яке приймають значення **М1 і М2**. Запис **Z** містить фіксовану частину (поле **N**) і варіантну з двома списками імен полів.

Опис запису має вигляд:

Type **T = (M1, M2);**

Var **TIP : T;**

**Z :** RECORD

**N :** integer**; {фіксована частина}**

CASE **TIP** of **{варіантна частина}**

**M1 : (список полів 1);**

**М2 : (список полів 2);**

End**;**

Таким чином:

1. початок варіантного запису відмічається службовим словом CASE, після визначення поля признаку вибору варіантів записується службове слово of.
2. варіантна частина повинна знаходитись після фіксованої наприкінці запису перед службовим словом END, тобто тільки одна варіантна частина.

Приклад 4**:** варіанти характеристики систем зброї

|  |  |  |
| --- | --- | --- |
| Тип зброї | Індекс | Маса |
| W (ZRK, RLS) | I | M |

Для ZRK:

|  |  |
| --- | --- |
| Дальність дії | Висота дії |
| D | H |

Для RLS:

|  |  |  |
| --- | --- | --- |
| Потужність передавача | Довжина хвилі | Спосіб обзору |
| P | L | SZ |

Опис запису:

Type **SS = (neprer, impuls);**

**TW = (ZRK, RLS);**

**Var WW : TW;**

**W :** RECORD

**I :** integer**;**

**M :** real**;**

CASE **WW of**

**ZRK : (D, H :** real**);**

**RLS : (P, L:** real**; SZ : SS);**

End**;**

Якщо тепер в програмі виконати оператор :

**WW :=ZRK;**

то далі можна використовувати оператори:

**W.I :=124;with w do**

**W.M :=15;⬄I := 124;**

**W.D :=40;M := 15;**

**W.H :=24;D := 40;**

**H := 24;**

Якщо виконано оператор

**WW := RLS;**

with **w** do

**I := 203;**

**M := 18;**

**P := 10;**

**L := 0.3;**

**SZ := impuls;**

Особливості:

1. для розміщення змінної комбінованого типу завжди відводиться фіксований об’єм пам’яті, який визначається найбільшим варіантом. Таким чином, різні варіанти одного запису якби “накладаються” один на одного у пам’яті, тобто займають одну і ту ж область.
2. мова Паскаль не має засобів контролю за правильністю роботи з варіантами, тому оператор повинен складати програму дуже коректно.

Синтаксичні діаграми:

1. комбінований тип:

record **-> <список полів>** -> end;

1. список полів:

**<фіксована частина> -> ;-> <варіантна частина> ->;**

1. фіксована частина:

**<індентифікатор> -> : -> <тип> -> ;**

1. варіантна частина:

case **-> <ідентифікатор> ->** of **-> <альтернатива> -> ;**

1. дискриминант:

**<ідентифікатор> -> : -> <ідентифік. типа>**

1. альтернатива:

**<константа> -> : -> <список полів> ->**

**4. Множини**

Цей тип даних використовують не так часто (одна з причин – низька швидкість виконання операції з множинами). Однак у ряді випадків його використання надає компактність і наочність програм.

Під множиною в ТР розуміють неупорядковану сукупність елементів, як це прийнято в математиці. Елементами множини можуть бути тільки змінні простих типів – скалярного та обмеженого.

Наприклад, цифри та символи латинського алфавіту:

[1, 3, 5, 7, 9,] – множина непарних чисел;

[‘A’, ’X’] – множина елементів типу char**;**

[ ] – порожня множина.

Опис множини:

1. Type

в 2-х розділах;

var

1. var– в 1-му розділі.

Для опису використовують таку форму:

**<ім’я змінної множини> :** SET OF **<тип елементів> ;**

**Приклад 1:**

Var

**M1 :** SET OF **‘A’ .. ’Z’;**

**M2 :** SET OF **1 .. 2;**

В опису можна вказуватине тільки границі значень, а також їх перелік:

**Приклад2:**

**M3 :** SET OF **‘A’ , ’R’ , ’C’;**

Для присвоювання імені множини змінних значень використовують конструкцію вида:

**<ім’я змінної множини> := <список>;**

Приклад3:

**M1 := [‘A’ , ‘B’ , ‘K’] ;**

**M2 := [2 , 3] ;**

тобто змінна **М1** приймає значення множини із елементів ‘A’,’B’,’K’, змінна M2 – із елементів 2 і 3.

По суті конструктор є перетворювач змінних простого типу у тип множини.

Над множинами виконують теоретико-множинні операції та операції відношень.

Теоретико-множинні операції:

1) поєднання “+”;

2) перетин “\*”;

3) різниця “-”.

Операції відношень:

1. належність елемента множині;
2. рівність, нерівність;
3. належність множини множині.

Приклади:

1. поєднання “+”;

А+В АВ

**A := [‘A’ , ‘B’];**

**B := [‘A’ , ‘K’];**

**A := A+B**

**{A = [‘A’ , ‘B’ , ‘K’]}**

1. перетин: “\*”

A\*B B

**A := [‘A’ , ‘B’];**

**B := [‘A’ , ‘K’];**

**S := A\*B {S=[‘A’]}**

1. різниця – формується множина з елементів першої множини, які не входять в другу множину

**R := A-B {R=[‘B’]}**

**A-B (A/B)**

Пріоритет виконання операцій:

( ) ⇒ \* ⇒ + ⇒ −

1. належність елемента множині – призначається службовим словом IN:

**<значення>** in **<множина>**

результат: TRUE- значення входить до множини;

FALSE- не входить.

Приклад:

**2** in **[1 … 10, 12] {**TRUE**}**

**5** in **[1, 2 ,7, 10] {**FALSE**}**

Цю операцію використовують для спрощення перевірок.ї

Наприклад:

if **(ch = ‘a’)** or **(ch = ‘b’)** or **(ch = ‘k’)** or **(ch = ‘y’)** then **S ,** або інакше

if **ch** in **[‘a’,’b’,’x’,’y’]** then **S**

1. перевірка на рівність, нерівність:

“=” – рівність 2-х множин;

“<>” – нерівність 2-х множин;

1. належність множин:

“<=” – перевірка належності множини лівого операнда правому;

“=>” - перевірка належності множини правого операнда лівому;

результат операції є :

TRUEабо FALSE**.**

Приклади:

Нехай задані множини

**M :** set of **1 .. 10;**

**L :** set of **‘A’ .. ‘D’;**

і виконані оператори присвоювання

**M := [2, 3, 4, 6];**

**L := [‘A’, ‘B’];**

тоді

**3** in **M {true}**

**[2, 4] <= M{true}**

**[ ] = L{false}**

**[‘A’] <> L{true}**

**‘E’** in **L{false}**

**([4] <= M)** and **(L =B) {false}**

**B** in **(M + [7, 8]){false}**

Особливість введення та виведення множини

Ввести та вивести множини за допомогою операторів **read** та **write** не можливо. Тому існують спеціальні прийоми вводу і виводу окремих елементів множини.

1. Послідовне введення елементів множини. Можна виконати за допомогою циклічного оператора з перед умовою. Для фіксації кінця вводу елементи порівнюються із втраченою константою. Коли вони співпадають, ввід закінчується. Внутрішнім оператором циклу є конструктор, який формує множину.

Нехай ознакою закінченням буде крапка ‘.’.

**MN := [ ]; {конструктор порожня множина}**

**M := ‘ ‘; {можна присвоювати будь-який символ}**

While **M <> ‘ . ‘** do

Begin

Writeln **(“ввести елемент множини”);**

Read **(M);**

**MN := MN + [M];**

End**.**

1. вивід елементів множини можна виконувати шляхом перевірки наявності елемента в множині і його виводу, якщо він міститься в множині. Нехай елемента множини

**ZZ : set of 1 .. 30;**

Потрібно вивести на екран, тоді фрагмент циклічного перебору має вигляд:

**For i := 1 to 30 do**

**If i in ZZ then Writeln ( i );**

**5. Загальні відомості про процедури та функції**

В практиці програмування часто зустрічаються випадки, коли при виконанні програми треба проводити однакові обчислення при різних початкових даних. Для виключення повторень і спрощення програми ці обчислення можна виділити в самостійну частину програми, яка може використовуватись багаторазово. Така автономна частка програми (яка реалізує визначений алгоритм і припускає звертання до неї із різних часток загальної програми) називається підпрограмою.

Розміщення підпрограм у програмі схоже з ієрархічним принципом побудови файлової системи.

Позитивні сторони використання підпрограм:

1. реалізується метод структурного програмування (модульна побудова);
2. економія пам’яті (яка виділяється для зберігання змінних підпрограми тільки під час роботи підпрограми);
3. імена змінних в основній програмі і підпрограми незалежні (потр. змінна А програми і підпрограми можуть мати різні фізичні значення);
4. самостійний характер підпрограм дозволяє їм створення різними користувачами, що прискорює процес програмування.

В мові Паскаль виділяють два видм підпрограм:

* процедура (**PROCEDURE**);
* функція **(FUNCTION).**

Принципи побудови Паскаль програми, процедури і функції ідентичні і відрізняються тільки заголовками.

Форми заголовків:

**Program <ім’я>** (список параметрів)

**Procedure <ім’я>** (список параметрів)

**Function <ім’я>** (список параметрів)

Блоки можуть бути не доложеними або вкладенеми один в одного. Відповідно вони носять назву зовнішнього або внутрішнього (рис.1).

Розглянемо принцип блочної побудови програми на рис.2.

Program

-----------------

Розділ процедур і функцій

**Begin**

**………**

**end**

Procedure

**Begin**

**……..**

**end.**

Procedure

**Begin**

**…………..**

**end.**

Рис. 2.

Із рис.2 видно, що процедури і функції розміщуються в останньому розділі блоку опису програми. Таких процедур і функцій може бути декілька і розміщуються вони в довільному порядку. В свою чергу, в їх розділах опису можна розміщувати процедури і функції 2-го рівня і т.д.. Наприкінці процедури і функції після **end** стоїть “ ; ”.

В зв’язку з тим, що кожна процедура або функція може містити свій розділ опису, виникає так звана проблема локалізації імен змінних.

Змінні і константи можна описувати у зовнішніх або внутрішніх блоках. Тому виникає два поняття змінних і констант: локальні і глобальні.

Локальні – існують у рамках того блока, де вони описані. Тобто за межами даного блока їх використовувати не можливо.

Глобальні – описані у зовнішньому блоці але використовують у внутрішньому блоці.

Наявність локальних і глобальних змінних дозволяє при творені складних програм використовувати одні і ті ж імена в різних блоках, що полегшує взаємодію програмістів. З іншого боку, можлива поява негативних побічних явищ, тому треба коректно створювати програми і придержуватись правил локалізації змінних.

Правила локалізації змінних:

1. змінна (константа) існує в рамках того блока де вона описана;
2. кожна змінна в межах блока повинна бути описана тільки один раз;
3. якщо у внутрішньому блоці використовується змінна, яка описана у зовнішніх блоках, то діє тільки опис самого внутрішнього блоку (найблищого);
4. якщо змінна описана як у внутрішньому так і у зовнішньому блоках, то при переході до внутрішнього блока опис зовнішнього блока не використовується, тобто діє опис внутрішнього блоку.

При переході до зовнішнього блока знову починається дія опису зовнішнього блока.

В мові Паскаль існує два типа процедур і функцій:

* з параметрами;
* без параметрів.

Вони відрізняються способом їх вмикання в основну програму.

**6. Процедури з параметрами**

Для успішної побудови блочних програм необхідно оволодіти двома методиками:

* розробки побудови процедури;
* використання процедури в програмі або в іншій процедурі.
  1. Методика розробки процедури.

Як було сказано, процедура складається із заголовка і блока. Блок будується стандартно і не відрізняється від блока програми за винятком того, що наприкінці процедури ставиться крапка з комою.

Розглянемо заголовок (рис.1). ім’я вибирають довільно. Параметри, які містяться у списку параметрів, носять назву формальних – т.б. вони не приймають ніяких значень, а тільки позначають місця, на які далі будуть поставлені так звані фактичні параметри.

Параметри позначають початкові дані (аргументи) і результати виконання процедури.

Список параметрів складається із окремих секцій, які відокремлюють крапкою з комою ( ; ).

В секції вказується перелік (список) параметрів і через двокрапку їх тип. Наприклад:

**Procedure R(x, y : real; k : integer; var s : real);**

Тут параметри **x** і **y** вказують першу секцію, **k –** другу, **s** – третю. Розміщення секцій параметрів в них довільне. В даному прикладі **x, y, k** – параметри значення; **s** – оператор-змінна, перед яким повинно стояти службове слово **var**.

Особливості цих параметрів ми розглянемо далі, а поки будемо вважати , що параметри-значення – аргументи функції, а параметри-змінні – результати виконання процедури.

Приклад1 розробки процедури:

**Procedure Sum (a, b : real; var y, z : real);**

**const n=10;**

**var i : integer;**

**begin**

**for i := 1 to n do**

**begin**

**y := a + b \* i;**

**z := 4 \* y – i;**

**end;**

**end.**

Тут **a** і **b** – вхідні параметри, їх значення передаються із основної програми в цю процедуру. Результати процедури (вхідні параметри)є обчислені значення **y** і **z**, які передаються із процедури в основну програму і там можуть бути використані.

* 1. Методика використання процедури у зовнішньому блоці.

Для використання процедур існує так званий оператор процедури:

**<ім’я процедури>\_ (**список фактичних параметрів**)**

Ім’я повинноспівпадати з іменем записаної процедури. Параметри у дужках носять назву фактичні. Вони відповідають реальним змінним, над якими виконують операції. Фактичні і формальні параметри повинні відповідати одне одному за містом їх розміщення і типом. Список фактичних параметрів не містить типів даних, бо вони вже вказані у списку формальних параметрів.

**Program PR;**

**…**

**procedure P(x, y : real; var z : real );**

**…**

**end;**

**begin**

**…**

**P(a, b, c);**{виклик процедури}

**…**

**end.**

Рис.3

Приклад 2: в програмі визначити довжину кола **C** і площі круга **S**, обмеженого цим колом; відстань **L** центра кола від початку координат **О**. Координати центра кола – **x, y** , радіус кола – **R**.

**Program KOLO ( input, output );**

**Var R : real; {радіус}**

**x, y : real:{координати центра}**

**C : real;{довжина кола}**

**S : real;{площа круга}**

**L : real;{відстань від центру}**

**Procedure KRUG ( R, x, y : real; var C, S, L : real);**

**Const pi = 3.14;**

**Begin**

**C := 2 \* pi \* R;**

**S := pi\* sqr ( R );**

**L := sqrt ( sqr(x) + sqr(y));**

**End;**

**Begin**

**Writeln ( ‘ ввести R, x , y :‘);**

**Read ( x, y , R);**

**KRUG ( R, x ,y, C, S, L);**

**Writeln (‘довжина кола = ’, C : 6 : 2);**

**Writeln (‘площа кола = ’, S : 6 : 2);**

**Writeln (‘відстань = ’, L : 6 : 2);**

**End.**

Результат виконання:

Ввести значення R, x, y:

386

довжина кола = 18,84

площа круга = 8,26

відстань = 10,00

Пояснення процесу звертання до процедур:

При виконанні дій, заданих оператором процедури, виконується передача фактичних параметрів, т.б. їх підставка на місце фіктивних. Таким чином в тілі процедури кожен формальний параметр заміняється фактичним. Виникає, так зване, модифіковане тіло процедури, яке безпосередньо реалізується ЕОМ. Після виконання процедури в комірках пам’яті ЕОМ залишаються результати, які поіменовані фактичними параметрами, які відповідають даним результатам.

Суть (трактовка) параметрів-значень і параметрів змінних.

Розглянемо особливості передачі параметрів із зовнішнього блоку до внутрішнього.

1-й спосіб – підстановка на місце формальних параметрів обчислених значень фактичних параметрів. Наприклад:

**Sum (3.5, 7.6, y, z);**

2-й спосіб – підстановка замість формальних параметрів імен фактичних параметрів:

**Sum (x1, x2, y, z);**

При цьому результати процедури завжди утримують імена (і не обов’язкого співпадаючи з іменами формальних параметрів). Наприклад:

**Sum (x1, x2, С, D);**

Розглянемо можливі варіанти опису фактичних параметрів у зовнішньому блоці і відповідне звертання до процедури.

Таблиця 1

І. Procedure P (x : тип; y : тип; var z : тип);

|  |  |  |
| --- | --- | --- |
| 1 | A – const; B - const;  C – var | P (A, B, C) |
| 2 | A – var; B – var;  C – var | A := F1; B := F2;  …………….  P (A, B, C); або  read (A, B);  ……………  P (A, B, C); |
| 3 | C – var; | P (3, 4 ,C); |

ІІ. Procedure P1 (var x : тип; var y : тип; var z : тип);

|  |  |  |
| --- | --- | --- |
| 4 | 1. var; 2. var; 3. var; | A := F1; B :=F2;  aбо  read (A, B);  ……………………  P1 (A, B, C); |

Коментарій табл.1:

1) фактичні параметри A,B у зовнішньому блоці описані як **const** і замінюють формальні параметри-значення **x, y** процедури **Р**;

2) фактичні параметри А і В обчислюються або вводяться у зовнішньому блоці перед вставленням їх у процедуру;

1. фактичні параметри А і В не об’явлені. Тому замість їх імен вставленні конкретні значення 3 і 4;
2. всі параметри задані змінні, тому при формуванні модифікованого тіла процедури на місце формальних параметрів підставлені імена фактичних параметрів.

**7. Процедури без параметрів**

В таких процедурах замість формальних параметрів використовують глобальні змінні.

Використання процедур без параметрів потребує уваги і коректності при складанні програм, точного дотримання правил локалізації змінних.

Для приклада 2 – програма обчислення параметрів кола при використані процедури без параметрів буде мати вигляд:

Приклад 2а:

**Program KOLO1 ( input, output );**

**Var R, x, y, С, S, L : real;**

**Procedure KRUG ( R, x, y : real; var C, S, L : real);**

**Const pi = 3.14;**

**Begin**

**C := 2 \* pi \* R;**

**S := pi\* sqr ( R );**

**L := sqrt ( sqr(x) + sqr(y));**

**End;**

**Begin**

**Writeln ( ‘ ввести R, x , y :‘);**

**Read ( x, y , R);**

**KRUG ;**

**Writeln (‘довжина кола = ’, C : 6 : 2);**

**Writeln (‘площа кола = ’, S : 6 : 2);**

**Writeln (‘відстань = ’, L : 6 : 2);**

**End.**

Змінні, яки використані у процедурі описані як глобальні.

**8. Особливості використання функцій**

За своїм призначенням і способом побудови функція мало чим відрізняється від процедур.

Особливості функцій:

1. має тільки один результат виконання;
2. результат позначається іменем функції і передається в основну програму, тому імені функції присвоюють необхідний тип даних:
3. форма побудови функції відрізняється від процедури тільки заголовком;
4. можна викликати в середині виразу:

**F := FAST (M) – FAST (K);**

Приклади заголовків функції:

1. **FUNCTION f (a, b :real) : real;**

А і В – дійсні параметри значення;

F – ім’я функції, яка отримає результат виконання функції типа real/

1. **FUNCTION f1 (x : real; var y, z : integer) : real;**

x - дійсні параметри значення;

y, z – цілі параметри змінні;

Суть параметрів-значень і параметрів-змінних аналогічна процедурам

1. **FUNCTION f2 : real;**

без параметрів

Приклад 3; знайти різницю факторіалів F=m! – k!

**FUNCTION FACT ( n: real) : real;**

**Var i :integer:**

**p : real;**

**Begin**

**p :=1:**

**for i := 1 to n do**

**p := p \* i;**

**FACT := p;**

**End;**

Тут **FACT** – ім’я функції, якому присвоюється результат типа real. Формальним параметром є **n** – параметр-значення типу integer.

(наприк. m i k):

**F := FAST (M) – FAST (K);**

Таким чином, якщо процедури і функції розробляються для багаторазового використання, краще їх використовувати з використанням параметрів, щоб досягнути максимальної незалежності від тієї програми, де вона використовується.

1. **Методи звертань до процедур і функцій**

Існує 3 варіанти звертання до процедур і функцій:

1. із зовнішнього блоку (цей спосіб розглянули у попередніх питаннях лекції);
2. із сусідньої функції;
3. із самої функції – рекурсія.

Розглянемо на прикладах 3-й і 2-й варіанти звертання:

Функцій (процедура), яка викликає сама себе називається рекурсивною.

Суть рекурсії розглянемо на прикладі4:

Визначити факторіал у рекурсивній формі:

n!=1, при n=0ж

n!=n(n-1)!, при n>0.

Тут n! визначається через (n!-1), т.б. через ту ж саму функцію.

**Programa FACT (input, output);**

**Var n : integer;**

**FUNCTION F ( v : integer) : integer;**

**Begin**

**If v = 0 then**

**F := 1**

**else**

**F := v \* F(v-1);**

**End;**

**Begin**

**Writeln (‘ ввести аргумент n ‘);**

**Read ( n );**

**If n<0 then**

**Writeln (‘факторіал не визнач.’);**

**else Writeln (‘факторіал’, n ,’= ’, F(n));**

**end.**

В прикладі4 рекурсивна функція **F** можно представити і не в рекурсивною:

**FUNCTION F( v : integer) : integer;**

**Begin**

**F := 1:**

**for i := 1 to v do**

**F := F \* i;**

**End;**

Особливість: на реалізацію рекурсивної функції ЕОМ затрачує значний час, хоча її опис більш компактний і наочний. Не рекурсивні функції обчислюються швидко.

Формально у мові Паскаль не дозволяється використовувати конструкції, які ще не описані.

Для виключення таких ситуацій використовують, так званий, випереджальний опис процедур і функцій. При цьому до опису процедури F1 вказується на подальші використання процедури F2. Тоді звертання з F1 до F2 допустимо (рис.5).

**Program Pr…;**

**Procedure F2 (…);**

**FORWARD;**

**Procedure F1 (…);**

**…**

**F2 (…);**

**End;**

**Procedure F2;**

**…**

**end;**

**begin**

**…**

**end.**

Рис.5

**Pr** містить 2-і незалежні процедури F1, F2. В F1 є звертання до F2, тому використаний випереджальний опис процедури F2, який складається із заголовка з параметрами і службового слова FORWARD.

Тоді сам опис F2 має заголовок без повторення формальних параметрів.

Особливості:

1. при наявності у складі програми декількох процедур або функцій, які звертаються одна до одної, то виникає, так звана, пряма рекурсія. У цьому випадку використовують випереджальний опис процедур.
2. Вхід до блоку процедур або функцій тільки через його початок, а вихід по закінченню всіх дій.

Т. ч. оператори **go to** – заборонені, т.б. всі мітки у блоці належать блоку і описуються у блоці. Поняття локальних і глобальних міток не існує.

1. при необхідності із середини блока можна використати стандартну процедуру **EXIT** , яка має форму:

**EXIT (аргумент)**

Якщо аргументом є ім’я програми або службове слово **PROGRAM** , то виконання програми достроково завершується.

Якщо аргументом є процедура або функція, то вони завершуються і управління передається в той блок, який викликав процедуру або функцію. Т.б. оператору, який стоїть після оператору виклику при використанні в процедурах і функціях глобальних змінних, можлива поява побічних ефектів.

**Висновки**

Таким чином, в Паскалі є широкі можливості використання процедур і функцій, що надає більшу наочність, компактність, структурованість програм; виникають умови для створення складних програмних комплексів групами програмістів; процес програмування стає більш систематичним і цілеспрямованим.