Лабораторна робота №4

Регулярні типи данних. Використання циклів

Мета роботи: здобуття практичних навичок використання регулярних типів даних -масивів та операторів циклу

Теоретичні відомості.

4.1. Масиви.

Для зберігання в пам'яті комп'ютера та обробки послідовностей однотипної інформації використовують масиви. Масив - це структурований тип даних, який складається з фіксованої кількості елементів однакового типу, який називається базовим. Базовим може бути будь-який раніше описаний тип, в тому числі структурований.

Елементи масиву можуть бути будь-якого типу. Індекси можуть бути тільки цілих, символьного та булевого типу, або обмежених типів, побудованих на їх основі. Найчастіше використовуються в якості індексного типу відрізки цілих типів.

Якщо при описанні масиву використовується один індекс, масив називається одновимірним, якщо два індекси - двовимірним, якщо n індексів - n-вимірним. Розмірність масиву обмежується лише об'ємом пам'яті комп'ютера.

4.1.1. Масиви в Pascal

Кількість елементів масиву в TP програмі визначається при описуванні і в процесі виконання програми не змінюється.

Опис масивів проводиться або як опис типу, або як опис змінної за допомогою ключового слова **`Array`**.

Приклад:

```
type     Arr = Array[1..20] of Integer;
```

```
var      A, B : Arr;
         C: Array[1..3,1..5] of Real;
         D: Array[3..8] of Arr;
         E: Array[byte] of ShortInt;
```

Формат опису за допомогою **Array** такий:

```
Array [ I_type ] of E_type;
```

де **I_type** - тип індексів;

E_type - тип елементів.

Робота з індексами масивів потребує підвищенної акуратності і розуміння принципів розташування елементів масиву в пам'яті комп'ютера. Елементи масиву займають суміжні комірки пам'яті, причому місце розташування потрібної комірки визначається з базової адреси (що обумовлено описом масиву) та зміщення (що обумовлено конкретним набором індексів), тому неправильний набір індексів може не викликати повідомлення про помилку, але призвести до помилки обчислень.

Наприклад:

Задано масив **A : Array [1..3,1..7] of integer**, який може бути представлений таблицею:

```
10   11   12   13   14   15   16
17   18   19   20   21   22   23
24   25   26   27   28   29   30
```

Далі в комірках пам'яті можебути, наприклад, така послідовність цілих чисел: **45, 46, 47, 48, 49**.

Звертання до масиву **A** за індексом **[2,4]** поверне значення **A[2,4]=20**. Якщо переплутати черговість індексів, то результат може бути, наприиклад, **A[4,2]=46** - значення, яке не має ніякого відношення до описаного масиву.

Автоматичний контроль знаходження індексу в заданних межах може бути включений директивою компілятору **{$R+}**, що призведе до генерації помилки при виході за межі. Однак використання такої директиви може бути виправдане тільки на етапі відлагоджування програми, оскільки викликає включення в результуючий код великої кількості умовних опреаторів, що призводить до суттєвого збільшення довжини та часу виконання програми.

В TP є можливість одним оператором присвоювання передати значення елементів одного масиву іншому масиву того ж типу. Так для описаних раніше масивів

```
type     Arr = Array[1..20] of Integer;
var      A, B : Arr;
```

можна провести операцію присвоювання **B:=A;.**

Після оголошення (опису) масиву кожен його елемент можна використовувати у лівій та правій частинах операторів присвоювання, як аргумент функцій та процедур та ін., для чого вказується його ім'я та індекс елемента в квадратних дужках.

Free Pascal має ще одну можливість – використання динамічних масивів. Розмір (кількість елементів) динамічного масиву може задаватись всередині програми, і навіть змінюватись декілька разів за час роботи програми. Це дозволяє, наприклад, спочатку організувати діалог для введення кількісного показника (розмір вибірки, розмір системи рівнянь, степінь поліному) а потім виділяти необхідну для розміщення масиву (масивів) пам’ять.

Для організації динамічних масивів застосовується ті ж прийоми, що й для статичних масивів, але без вказування розмірності:

```
type     MyT = Array of Real;
         MyTT = Array of Array of integer
var      A : MyT;
         B: Array of Integer;
         C: MyTT;
```

У ході виконання програми можна встановити розмір масиву, використовуючи оператор **SetLength:**

```
SetLength(A, N);
SetLength(B, 8);
SetLength(C, 10, 10);
```

4.1.2. Масиви в VBA

На відміну від скалярних типів даних, які у VBA можуть використовуватись без опису, масиви необхідно описувати завжди. В програмі можна описати і використовувати 2 види масивів: фіксовані – розмір яких не змінюється під час роботи програми та динамічні - розмір яких можна змінювати.

Опис масивів, як і інших змінних, може знаходитись у будь-якому місці проограми до їх першого використання. Опис здійснюється оператором **Dim**.

Dim Aff(20) as Integer - описує масив елементів типу **Integer**.

Dim Baa(15) as Single - описує масив елементів типу **Single**.

Dim Mat(10, 10) as Double – описує двовимірний масив елементів типу **Double**.

Як можна помітити, VBA та TP використовують різну систему індексації масивів. В TP при описі вказують, фактично, усі можливі значення для кожного індексу, в VBA записується тільки верхня межа для відповідних індексів.

Якщо в програмі за допомогою оператора **Option Base** не вказано іншого, то індексація будь-якого масиву починається з нуля (**0**). Це значить, що масив **Aff**, описаний вище містить елементи з індексами від 0 до 20 (21 елемент). Якщо в програмі застосовано оператор **Option Base 1**, то той же масив буде містити елементи з індексами від 1 до 20 (20 елементів).

Для створення динамічного масиву застосовую формат:

```
Dim Agg() as Integer
```

Однак перед використанням його необхідно активувати, вказавши розмір:

```
ReDim Agg(10) as Integer
```

Перевага динамічного масиву полягає в тому, що оператор **ReDim** можна застосовувати щодо масиву під час виконання програми довільну кількість разів. Однак, необхідно пам’ятати, що кожне використання цього оператора обнуляє існуючі елементи масиву. Для збереження елементів масиву, при зміні його розміру використовують формат оператора:

```
ReDim Preserve Agg(10) as Integer.
```

Іншою особливістю оператора **ReDim** є те, що з його допомогою можна змінити розмірність масиву, як кількість елементів так і кількість вимірів, однак не можна змінювати тип елементів.

4.2. Організація циклів.

Циклічне виконання деякої сукупності наперед визначених дій є одним з основних прийомів програмування. Більшість мов програмування має

стандартний набір операторів організації циклів, які коротко можна описати як: цикли з лічильником, які виконуються наперед відому кількість разів; цикли з передумовою – виконуються тільки у випадку, якщо виконується умова; цикли з післяумовою – виконуються перший раз завжди, далі у випадку виконання умови.

4.2.1. Організація циклів в середовищі TP

Цикли з лічильником.

Оператор **For** використовується в тому випадку, якщо кількість повторень циклу відома до моменту його організації. Наприклад цикл **For** є одним з основних при обробці масивів данних.

Формат оператора:

```
For cnter:= StartValue To EndValue Do <Operator>,
```

або

```
For cnter:= StartValue DownTo EndValue Do <Operator>,
```

де **cnter** - параметр циклу (лічильник) – змінна одного з цілих типів;

StartValue - початкове значення лічильника;

EndValue - кінцеве значення лічильника.

<Operator> - оператор тіла циклу (в т.ч. складний оператор)

Перший варіант оператора (**For ... To ... Do**) відповідає випадку збільшення **StartValue**, а другий варіант (**For ... DownTo ... Do**) - випадку зменшення **StartValue**.

Якщо використано перший варіант, але **StartValue** більше **EndValue**, або навпаки - використано другий варіант оператора а **StartValue** менше **EndValue** то оператор, що складає тіло циклу не виконується. При виконанні циклу його параметр послідовно збільшується (**for .. to**) або зменшується (**for .. downto**) на одиницю при кожному повторі. В тілі циклу може бути використано інший (вкладений) оператор **for**. Кількість вкладень циклів теоретично не обмежується, а практично обмежена розміром пам'яті комп'ютера, що відводиться під стек.

Приклад:

```
var N, i : Word;
    A : Array[1..2,1..20] of Real;
BEGIN
    Write('Введіть кількість експериментальних точок
(до 20)');
    ReadLn(N);
    For i:=1 to N Do
        begin
        Write('Введіть X[',i,']');
        Read(A[1,i]);
        Write('Y[',i,']');
        ReadLn(A[2,i]);
    end;
END.
```

Наведений фрагмент програми забезпечує введення кількості та значень експериментальних точок X та Y. При використанні циклу **for** важливо не допустити зміни значення параметру циклу всередині тіла циклу.

Оператор While - оператор циклу з передумовою.

Формат:

While <boolExp> Do <Operator>

<boolExp> - вираз, який повинен мати результат булевого типу (**True** або **False**);

<Operator> - оператор, у тому числі і складний, виконується тільки в тому випадку якщо **<boolExp> = True**.

Приклад:

```
While Ch = ' ' do Ch := GetChar;
```

Рядок в програмі чекає вводу з клавіатури символа, що відмінний від пропуску.

```
i:= 1;
While M <= Sqr(i) Do
Begin
    If M = Sqr(i) then
    writeln('число ', M, ' є квадратом числа ', i)
    Inc(i);
End;
```

Фрагмент програми шукає натуральне число, що при зведенні до квадрату дасть задане.

Оператор Repeat - оператор циклу з післяумовою.

Формат:

```
Repeat
<Operator>   {тіло}
<Operator>   {циклу}
...
Until <boolExp>
```

Оператори, що складають тіло циклу виконуються до тих пір, поки значення **<boolExp> = False**. Перевірка істиності виразу проводиться після виконання тіла циклу.

Приклад:

```
repeat Ch := GetChar until Ch <> ' ';
repeat
    Write('Enter value: ');
    ReadLn(I);
until (I >= 0) and (I <= '9');
```

4.2.2. Організація циклів в середовищі VBA

Цикли з лічильником

Як і в ТВ, у VBA цикл з лічильником реалізуєть у вигляді циклу **For**, який, проте, має дещо інший синтаксис.

Формат оператора для створення циклу наступний:

```
For counter = StartValue To EndValue [Step Stp]
<Operator>
Next counter
```

де **counter** - параметр циклу (лічильник);

StartValue - початкове значення лічильника;

EndValue - кінцеве значення лічильника.

<Operator> - оператор тіла циклу

Stp – довільне число.

Якщо при створенні циклу слово **Step** з наступним за ним числом опущене, то оператор працює, вважаючи крок зміни лічильника рівним одиниці. У випадку явного задання величини кроку, оператор циклу використовує його. Число **Stp** може бути як цілим так і дійсним, як позитивним так і негативним.

Приклади застосування:

```
For i=1 to 10
A(i)= Sheets("Sheet1").Cells(0+i,1).Value
A(i)=A(i)+5
Sheets("Sheet1").Cells(0+i,1).Value=A(i)
Next i

For i=2.3 to 4.2 step 0.4
MsgBox(i)
Next i
```

Цикли Do … Loop

Цикли типу **Do** … **Loop** в VBA можуть використовуватись для організації циклів як з передумовою так і з післяумовою. Для встановлення умови можуть використовуватись слова **Until** (поки не) та **While** (поки):

Цикл з передумовою:

```
x = 0
Do Until x = 100
x = x + 1
Loop
```

Цикл з післяумовою:

```
x = 0 Do
x = x + 1 Loop Until x = 100
```

Цикли While … Wend

Цикл типу **While**…**Wend** являються циклами з передумовою. Наприклад:

```
x = 0
While x < 50
x = x + 1
Wend
```

Завдання на лабораторну роботу:

Скласти та відлагодити програму, яка дозволить вводити значення елементів двох матриць, та знаходити їх добуток, або іншу за вказівкою викладача. Використати різні способи організації циклів. Зробити висновки.

Рекомендована література: [1], [2], [4], [5], [6] , [7], [8]