ЛАБОРАТОРНАЯ РАБОТА

Тема: "Процедуры и функции"

Цель работы

1. Приобретение практических навыков в программировании процедур и функций.
2. Изучение механизма передачи параметров.
3. Знакомство с локальными и глобальными переменными.

Краткие сведения из теории

Процедуры и функции являются мощным средством языка программирования. Это средство удобно применять в тех случаях, когда при решении задачи возникает необходимость в программе некоторую совокупность операторов повторять несколько раз. Так например, может возникнуть необходимость один и тот же цикл использовать в нескольких местах программы.

Функцию или процедуру можно сравнить с мини-программой, именно поэтому их называют иногда одним общим именем - "подпрограмма (ПП)". ПП оформляется подобно программе: в начале записывается заголовок ПП, затем следует декларативная часть ПП и после процедурная. В декларативной части описываются все данные, область действия которых ограничена телом данной ПП. Эти данные называются локальными. Данные, объявленные в основной (главной) программе, называются глобальными и они могут использоваться в любой ПП, входящей в основную программу. В процедурной части описывается тело ПП, реализующее алгоритм решения, и которое заключается в операторные скобки BEGIN, END.

ПП помещается сразу же после объявления всех переменных. Заголовок ПП для подпрограмм-функций начинается с ключевого слова FUNCTION, для подпрограмм-процедур с ключевого слова PROCEDURE. Эти ключевые слова играют роль признаков, которые распознает компилятор. Так как ПП исполняется не сразу и возможно не один раз, то компилятор, встретив тело ПП, должен его пропустить.

11.1. Функции

Общая схема функции следующая:

FUNCTION < идентификатор функции >(<параметр: тип параметра> [,<параметр: тип параметра>, ...]) : <тип результата функции>;

< декларативная часть (разделы LABEL, CONST, TYPE, VAR) -объявление локальных данных >

BEGIN

< процедурная часть функции - тело функции >

END;

В теле основной программы функция вызывается по имени, это значит, что после FUNCTION необходимо записать идентификатор функции, а затем в круглых скобках перечислить все параметры функции. Так как язык Паскаль сильно типизирован, то он требует, чтобы после каждого параметра был указан его тип. Результатом вычисления функции всегда получается одно значение. Поэтому после круглых скобок в заголовке функции необходимо указать тип результата, вычисляемого функцией.

Ко всем функциям обращаются одинаково: в любом предложении программы они играют роль переменной.

Самым простым и наглядным примером использования функций являются стандартные функции, например, функция LENGTH(St). Эта функция может применяться в программе всякий раз, когда необходимо вычислить длину стринга, в данном случае стринга St. Все стандартные функции входят в состав компилятора, то есть они описаны в теле самого компилятора. LENGTH - это идентификатор функции, St - аргумент функции. Заголовок этой функции может быть следующий:

```
FUNCTION Lenght(St : string) : byte;
```
Тип результата, возвращаемого этой функцией, BYTE.

Пример.

Напишем подпрограмму, позволяющую вычислять степенную функцию a^n, причем n может иметь только положительные целые значения.

В этом примере параметрами функции должны быть: основание и степень.

```
        { вычисление степенной функции }
FUNCTION Degree(Base: real; Power: integer) : real;
        { обьявление локальных переменных i, Y
        i - параметр цикла, считающий степени;
        Y - промежуточная переменная, содержащая текущие значения степенной функции }
VAR
     i : integer;
     Y : real;
        { тело функции или процедурная часть }
BEGIN
    Y:= 1;
    For i:= 1 to Power do
    Y:= Y * Base;
    Degree:= Y;
END;
```

В основной программе обращаться к этой подпрограмме (функции) можно многократно, например:

```
    . . .
VAR
     B, X, Z : real;
     P       : integer;
    . . .
 BEGIN { основная или главная программа }
     . . .
     B:= 1.5;  P:= 3;
     Z:= Degree(B, P); { основание = 1.5, степень = 3 }
     . . .
     WriteLn('Значение степенной функции = ', Degree(X*2, 10));
     . . .
 END.
```

Проследим работу функции по первому вызову. Значения переменных B и P: 1.5 и 3 передаются подпрограмме DEGREE. Однако после входа в подпрограмму мы эти переменные называем уже не B и P, а Base и Power. Имена Base и Power, фигурирующие в подпрограмме, являются просто "пустышками", замещаемыми при работе функции (вычислении) конкретными значениями. По терминологии ПП параметры, используемые в самой подпрограмме, называются формальными, а параметры, используемые в основной программе при ссылке к функции, называются фактическими. В нашем примере B и P - фактические параметры, Base и Power - формальные. Этот прием введения фактических и формальных параметров удобен тем, что при многократном вызове подпрограммы мы можем передавать различные параметры, как это сделано в представленном выше фрагменте программы:

Degree(B, P) и Degree(X*2, 10).

Имена формальных и фактических параметров не обязаны совпадать, хотя если случайно они и оказываются одинаковыми, никаких проблем не возникнет. Однако необходимо,

чтобы типы формального и фактического параметров были согласованы (например, оба были типа INTEGER или REAL).

При выполнении любой ПП в первую очередь происходит сопоставление формальных и фактических параметров. С этого момента ссылка на Base и Power в подпрограмме фактически означает обращение к B и P. Следовательно, команды, вычисляющие степенную функцию, используют значения B и P (1.5 и 3). Результат вычислений помещается в имя DEGREE, в этом и состоит механизм, посредством которого функция "посылает" свой ответ в основную программу. После возврата в основную программу результат, переданный функцией, назначается переменной Z. При этом значение, вычисленное функцией и помещаемое в DEGREE, имеет тот же тип, что и переменная Z.

Структуру программы, содержащей подпрограмму-функцию, можно представить следующим образом:

```
PROGRAM Main;
CONST       { декларативная
    . . . ;   часть
VAR           главной
    . . . ;   программы }
{ процедурная часть главной программы }
FUNCTION Sample( . . . ) : . . .; { заголовок функции }
VAR         { декларативная часть подпрограммы }
   . . . ;
{ процедурная часть ПП }
BEGIN   { Sample }
   . . .
   Sample:= . . . ;
END;    { Sample }
BEGIN   { главная программа }
   . . .
 < любое количество вызовов функции Sample >
   . . .
END.    { конец главной программы }
```

11.2. Процедуры

Подпрограмму следует оформить в виде процедуры, если она предназначена для решения задачи одного из двух типов. Задача первого типа: требуется выполнить некую последовательность действий, не возвращая результирующего значения. Задача второго типа: требуется изменить значения одного или нескольких фактических параметров.

Процедура, как и функция, помещается в конце декларативной части программы. Компилятор распознает процедуру по ключевому слову PROCEDURE. С этого слова начинается заголовок процедуры, после которого следует идентификатор процедуры, а затем в круглых скобках перечисляются формальные параметры процедуры.

В Паскале передать параметры подпрограмме можно двумя способами. До сих пор мы имели дело только с одним из них. Этот способ называют передачей параметров по значению. Он состоит в том, что значение фактического параметра назначается соответствующему формальному параметру. Другими словами, перед началом выполнения процедуры вычисляется конкретное значение фактического параметра (например, 1.5 или 3). Затем полученное значение копируется в соответствующий формальный параметр, принадлежащий процедуре. Как только начинается выполнение процедуры, никакие изменения значения формального параметра уже не оказывают влияния на значение соответствующего фактического параметра. Это значит, что по окончании работы процедуры фактический параметр будет иметь точно такое же значение, каким он обладал до начала работы процедуры, вне зависимости от того, что происходило с формальным параметром. Такой способ передачи параметров действует при работе с функциями.

Второй способ передачи параметров называется передачей параметров по ссылке или по адресу. При передаче параметра по ссылке в процедуру пересылается уже не значение аргумента, а его местоположение (адрес) в памяти компьютера. Чтобы сообщить компилятору о намерении передать такой параметр, в заголовке процедуры в списке формальных параметров следует указать слово VAR. Если формальный параметр снабжен атрибутом VAR, а соответствующий ему фактический параметр является переменной, то любые изменения формального параметра будут отражаться в значениях фактического параметра, поскольку теперь формальный и фактический параметры занимают одну и ту же область памяти.

Общая схема процедуры аналогична схеме функции со следующими изменениями:

ключевое слово FUNCTION заменяется на PROCEDURE;

отсутствует тип результата.

Вызов процедуры в основной программе оформляется как отдельное предложение, состоящее из имени процедуры и пары круглых скобок, в которых через запятую перечислены фактические параметры. Предложение заканчивается как обычно символом ";".

Рассмотрим пример на задачу первого типа: необходимо выполнить перемножение двух матриц A(3x4) и B(3x4).

Оформим в виде процедуры ввод матрицы. Программа в этом случае может иметь вид:

```
{************************************************************}
{*       программа перемножения двух матриц A, B            *}
{************************************************************}
Program Main;
Const
   N = 3; { количество строк    в матрице }
   M = 4; { количество столбцов в матрице }
Type
   MatrTyp = array[1..N, 1..M] of real;
Var
   A, B, C    : MatrTyp;  { массивы можно передавать только как
                            простой тип }
   i1, i2, i3 : byte;     { индексы массивов }


{************************************************************}
{*          процедура ввода значений матрицы                *}
{************************************************************}
Procedure GetMatr(NameMatr : string; var Matr : MatrTyp);
Var
   i, j : byte;
Begin     { GetMatr }
   WriteLn('Введите значения матрицы ', NameMatr);
   For i:= 1 to N do
   begin
     WriteLn('Строка ', i);
     For j:= 1 to M do
       Read(Matr[i,j]);
     WriteLn;     { перевод на новую строку }
   end;
End;      { GetMatr }

{************************************************************}
```

```
{*                          главная программа                          *}
{*****************************************************************}
Begin
   GetMatr('A', A); {Вызов процедуры для ввода значений матрицы A}
   GetMatr('B', B); {Вызов процедуры для ввода значений матрицы B}
   { алгоритм перемножения матриц }
   For i1:= 1 to N do
    For i2:= 1 to M do
    begin
      C[i1,i2]:= 0;
      For i3:= 1 to N do
        C[i1,i2]:= C[i1,i2] + A[i1,i2] * B[i3,i1];
    end;
   { вывод значений результирующей матрицы }
   For i1:= 1 to N do
   begin
    For i2:= 1 to M do
      Write(' C[',i1:1, ',', i2:1, '] = ', C[i1,i2]:8:3);
    WriteLn; { перевод на новую строку }
   end;
End.     { конец главной программы }
```

11.3. Различия между процедурами и функциями

Главное различие (из которого следуют все остальные) состоит в том, что функция всегда возвращает, причем в явной форме, одно-единственное значение, которое может быть использовано в качестве составной части выражения; процедура такого значения не возвращает. Однако применительно к процедуре все же можно говорить о возвращаемой информации - процедура способна изменять значения своих параметров (тех, что описаны с атрибутом VAR).

Помимо главного различия можно отметить ряд второстепенных различий синтаксического характера. Так, например, заголовок функции всегда завершается указанием типа возвращаемого значения. В заголовке процедуры такая информация не нужна. Для функции типично, чтобы в качестве последнего шага имени функции было назначено некоторое вычисленное значение. В процедурах этого нет. И наконец, еще одно различие. Поскольку функция возвращает какое-то значение, вызов функции может появляться прямо в выражении. Вызов процедуры не может быть частью выражения - это всегда отдельное предложение.

Контрольные вопросы

1. Для чего предназначены функции?
2. Для чего предназначены процедуры?
3. Чем отличаются формальные и фактические параметры?
4. Опишите способы передачи параметров в подпрограммы и их особенности?
4. Что включает в себя заголовок подпрограммы?
5. Чем отличаются глобальные и локальные переменные?
6. Какая разница между процедурой и функцией?

Задание к работе

1. Модифицируйте подпрограмму, вычисляющую степенную функцию так, чтобы она вычисляла и отрицательные степени.
2. Напишите подпрограмму, способную вычислять любые степени: положительные и отрицательные, целочисленные и действительные.
3. Выполните индивидуальное задание:

Дано несколько массивов чисел. Длины массивов заданы в варианте индивидуального задания. Требуется в каждом массиве найти наибольший и наименьший элементы и отобразить их на экране, затем все компоненты каждого массива возвести в квадрат и снова найти наибольший и наименьший элементы. Вычисление максимальной и минимальной величин оформить в виде процедуры, глобальные параметры в процедуре не использовать.

Методические указания

1. При выполнении пункта 2 задания необходимо использовать экспоненциальную и логарифмическую функции.
2. При выполнении пункта 3 задания необходимо знать, что:
   a) если в качестве исходной информации в процедуру передается массив, то его следует передавать по ссылке для экономии памяти, так как в этом случае при вызове процедуры не образуется локальный массив;
   b) несмотря на то, что обрабатываемые массивы разной длины, они описываются в программе как массивы одного и того же типа, так как при обращении к процедуре типы соответствующих формальных и фактических параметров должны совпадать.
3. Составить алгоритм решения задачи.
4. Написать программу и откомпилировать ее.
5. Составить контрольный тест и отладить (протестировать) программу.
6. Составить отчет и представить его к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.
4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

Ввести и обработать:

1) два двумерных массива, содержащие соответственно 3x5 и 4x8 вещественных элементов;
2) три массива, содержащие соответственно 3, 6 и 8 целых элементов без знака;
3) четыре массива, содержащие соответственно 4, 6, 3 и 5 целых элементов со знаком;
4) два массива, содержащие соответственно 4 и 6 вещественных элементов;
5) три массива, содержащие соответственно 5, 10 и 4 целых элементов без знака;
6) четыре массива, содержащие соответственно 3, 5, 8 и 6 вещественных элементов;
7) два трехмерных массива, содержащие соответственно 2x3x2 и 3x4x2 вещественных элементов;
8) четыре массива, содержащие соответственно 4, 7, 3 и 5 вещественных элементов;
9) три двумерных массива, содержащие соответственно 2x5, 3x6 и 3x4 целых элементов без знака;
10) два двумерных массива, содержащие соответственно 6x2 и 3x2 вещественных элементов.