Методические указания

Лабораторная работа 9

Процедуры и функции. Рекурсия.

Рекурсия

Понятие рекурсии

Рекурсивным называется объект, который частично определяется через самого себя. В программировании ***рекурсия*** – способ организации вычислительного процесса, при котором процедура или функция **вызывает сама себя.**

Рекурсивные определения широко используются во многих областях, особенно в математике.

Рассмотрим функцию факториала ***n*!**. Как правило, ее определяют как произведение первых ***n*** целых чисел:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

Такое произведение можно вычислить с использованием итеративной конструкции цикла (это решение приводилось ранее, см. занятие 4):

```
program FactorialCycle;
var  n, i: integer;
     fact: longint;
begin
  write('Введите число n:');  readln(n);
  fact:=1;
  for i:=1 to n do fact:=fact*i;
  writeln('Факториал n!=',fact);
end.
```

Однако существует также другое (рекурсивное) определение факториала, в котором используется ***рекуррентная*** формула:

(1) **0! = 1**
(2) для любого $n > 0$ $n! = n \cdot (n - 1)!$

Этому определению факториала соответствует программа, использующая рекурсивную функцию, которая вызывает сама себя:

```
program FactorialRecursion;

function fact(i:integer):longint;
begin
  if i=1 then fact:=1   {условие останова}
         else fact:=i*fact(i-1);   {рекурсивный вызов}
end;

var  n:integer;
begin
```

```
  write('Введите число n:');  readln(n);
  writeln('Факториал n!=', fact(n));
end.
```

Следует обратить внимание, что здесь совсем **нет операторов цикла.** При этом операция умножения будет **повторяться *n*–1 раз,** поскольку функция `fact` вызывает саму себя *n*–1 раз.

Заметьте, что использование рекурсии позволяет легко (почти дословно) запрограммировать вычисления по рекуррентным формулам.

Разбор рекурсии на примере

Разберем подробнее приведенный пример рекурсивного вычисления факториала ***n*!**.

При каждом ***рекурсивном вызове*** (`fact(i-1)`) выполнение текущей подпрограммы приостанавливается, но ее переменные не удаляются из памяти. Происходит новый вызов подпрограммы, для переменных которой также выделятся память, и так далее. Образуется последовательность прерванных процессов, из которых выполняется всегда последний. Передаваемый в подпрограмму параметр каждый раз уменьшается на 1.

Когда параметр станет равным 1, выполнится ***условие останова*** (`i=1`). Рекурсивные вызовы закончатся, и функция вернет значение равное 1 (`fact:=1`). Текущий вызов подпрограммы окажется последним. Затем в обратном порядке последовательно закончат работу все вызванные подпрограммы.

Последняя завершенная функция (она же – первая вызванная) вернет результат, который будет выведен на экран командой `writeln`.

Глубина и уровень рекурсии

Максимальное число рекурсивных вызовов подпрограммы без возвратов называется ***глубиной*** рекурсии. Число рекурсивных вызовов в каждый конкретный момент времени называется ***текущим уровнем*** рекурсии.

В рассмотренном примере, если пользователь ввел число `n`=5, то общее количество вызовов функции `fact` равно 5, а глубина рекурсии (количество рекурсивных вызовов) равна 4. При первом вызове функции `fact` (в команде `writeln`) текущий уровень рекурсии считается равным нулю, т.к. функция еще не успела вызвать саму себя.

Поскольку имена параметров и локальных переменных не меняются при каждом рекурсивном вызове, то **воспользоваться значением некоторой переменной *i*-го уровня рекурсии можно, находясь только на этом *i*-м уровне.** В рассмотренном примере у функции `fact` нет локальных переменных, но есть параметр `i`, область видимости которого соответствует данному правилу.

Недостатки рекурсии

Для каждого нового рекурсивного вызова подпрограммы выделяется память, которая освобождается только после достижения всей глубины рекурсии. Поэтому выполнение рекурсивных процедур и функций требует **значительно большего объема оперативной памяти** во время выполнения программы, чем нерекурсивных.

Если глубина рекурсии очень велика, то это может привести к **переполнению памяти.**

К тому же рекурсивные алгоритмы выполняются **медленнее.**

Формы рекурсивных подпрограмм

Условие останова

Рекурсивное определение позволяет с помощью конечного выражения определить бесконечное множество объектов. А с помощью рекурсивного алгоритма можно определить **бесконечное вычисление,** причем алгоритм не будет содержать повторений фрагментов кода.

Рекурсивные подпрограммы, не содержащие ***условия останова*** – **прекращения рекурсивных вызовов,** – приводят к бесконечным процессам.

Например:

```
procedure endless;
  begin
    writeln('У попа была собака, он ее любил.');
    writeln('Она съела кусок мяса, он ее убил.');
    writeln('В землю закопал, надпись написал:');
    endless;
  end;
begin
  endless;
end.
```

Действие программы закончится, когда будет исчерпана вся свободная память. Использовать подобные рекурсивные процедуры и функции **невозможно** из-за ограниченности оперативной памяти.

Поэтому **главное требование** к рекурсивным подпрограммам заключается в том, что ***вызов рекурсивной подпрограммы должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным.***

Если это условие истинно, то ***рекурсивный спуск*** продолжается. А если условие становится ложным, то спуск заканчивается и начинается поочередный ***рекурсивный возврат*** из всех вызванных на данный момент копий рекурсивной подпрограммы.

Структуры рекурсивных подпрограмм

Структура рекурсивной подпрограммы может принимать **три** различные формы.

1) *Выполнение действий на рекурсивном спуске*

```
procedure rec;
  begin
    ...
    Операторы;
    ...
    if Условие then rec;
  end;
```

2) *Выполнение действий на рекурсивном возврате*

```
procedure rec;
  begin
    if Условие then rec;
    ...
    Операторы;
    ...
  end;
```

3) *Выполнение действий как на рекурсивном спуске, так и на рекурсивном возврате*

```
procedure rec;
  begin
    ...
    Операторы1;
    ...
    if Условие then rec;
    ...
    Операторы2;
    ...
  end;
```

или то же самое, но с проверкой условия вначале:

```
procedure rec;
  begin
    if Условие then
      begin
        ...
        Операторы1;
        ...
        rec;
        ...
        Операторы2;
        ...
      end;
  end;
```

Многие задачи безразличны к тому, какая форма рекурсивных подпрограмм используется. Но существуют классы задач, при решении которых требуется сознательно управлять ходом работы рекурсивных процедур и функций.

Таблица трассировки

Для демонстрации действий, выполняемых рекурсивной подпрограммой, используют ***таблицу трассировки*** значений ее параметров по уровням рекурсии.

В рассмотренном примере вычисления проводятся на рекурсивном возврате, и при вводе пользователем `n`=5 таблица трассировки:

Текущий уровень рекурсии	Рекурсивный спуск	Рекурсивный возврат
0	Ввод: `n=5  fact(5);`	Вывод: `n!=120`
1	`i=5  fact(4);`	`fact:=5*24 (=120);`
2	`i=4  fact(3);`	`fact:=4*6  (=24);`
3	`i=3  fact(2);`	`fact:=3*2  (=6);`
4	`i=2  fact(1);`	`fact:=2*1  (=2);`
5	`i=1  fact:=1;`	

Содержание