Занятие 3. Рекурсия и рекурсивные алгоритмы

1 Цель

Изучить принципы работы рекурсивных алгоритмов и научиться применить полученные занятия на практике.

2 Краткая теория

2.1 Понятие рекурсии

Рекурсией называется метод определения функции через её предыдущие и ранее определенные значения, а так же способ организации вычислений, при котором функция вызывает сама себя с другим аргументом.

Большинство современных языков высокого уровня поддерживают механизм рекурсивного вызова, когда функция, как элемент структуры языка программирования, возвращающая вычисленное значение по своему имени, может вызывать сама себя с другим аргументом. Эта возможность позволяет напрямую реализовывать вычисление рекурсивно определенных функций. При этом любой рекурсивный алгоритм может быть реализован итерационно (через циклы) и наоборот.

Рекурсивная программа не может вызывать себя до бесконечности, поэтому важным моментом реализации любой рекурсивной программы является наличие условия завершения, позволяющее программе прекратить вызывать себя.

Таким образом, рекурсия в программировании может быть определена как сведение задачи к такой же задаче, но уже более простой для решения (манипулирующей более простыми данными).

Как следствие, рекурсивная программа должна иметь как минимум два пути выполнения, один из которых предполагает рекурсивный вызов (случай «сложных» данных), а второй – без рекурсивного вызова (случай «простых» данных).

При рекурсивном погружении функция вызывает точно такой же новый экземпляр самой себя. При этом сама функция как бы еще не завершилась, а новый ее экземпляр уже начинает работать. И только когда новый экземпляр завершит работу (вернет результаты вычислений), будет продолжена работа самой функции. Информация о таких незавершенных вызовах рекурсивных подпрограмм (а это, в самом простом представлении, значения переменных, необходимых для работы подпрограммы) запоминается в специальной области памяти – стеке.

2.2 Пример рекурсивного метода

Процедура или функция может содержать вызов других процедур или функций. В том числе процедура может вызвать саму себя. Никакого

парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту процедуру. Без разницы, какая процедура дала команду это делать.

Пример 1.1 Пример рекурсивной процедуры

```
procedure Rec(a: integer);
begin
     if (a > 0) then
         Recursion(a-1);
     writeln(a);
 end;
```

Рассмотрим, что произойдет, если в основной программе поставить вызов, например, вида Rec(3). Ниже представлена блок-схема, показывающая последовательность выполнения операторов.

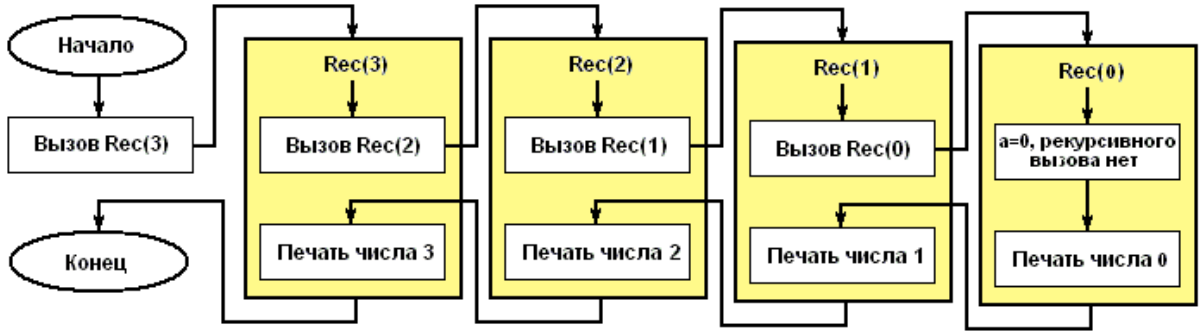

Рисунок 1.1 – Блок-схема работы рекурсивной процедуры

Процедура Recursion вызывается с параметром a = 3. В ней содержится вызов процедуры Recursion с параметром a = 2. Предыдущий вызов еще не завершился, поэтому можете представить себе, что создается еще одна процедура и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра процедуры. Количество одновременно выполняемых процедур называют **глубиной рекурсии**.

Четвертая вызванная процедура (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к процедуре, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все процедуры. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Еще один визуальный образ происходящего представлен на рис. 1.2.

Рисунок 1.2 – Схема работы рекурсивной процедуры

Выполнение процедуры Rec с параметром 3 состоит из выполнения процедуры Rec с параметром 2 и печати числа 3. В свою очередь выполнение процедуры Rec с параметром 2 состоит из выполнения процедуры Rec с параметром 1 и печати числа 2. И т. д.

> Подумайте, что получится при вызове описанной процедуры, если операторы if и writeln поменялись местами.

Обратите внимание, что в приведенных примерах рекурсивный вызов стоит внутри условного оператора. Это необходимое условие для того, чтобы рекурсия когда-нибудь закончилась. Также обратите внимание, что сама себя процедура вызывает с другим параметром, не таким, с каким была вызвана она сама. Если в процедуре не используются глобальные переменные, то это также необходимо, чтобы рекурсия не продолжалась до бесконечности.

2.3 Рекурсивные вычисления

Приведённый выше пример выводил результат на экран, однако чаще всего рекурсивные функции используются для организации вычислений, когда необходимо получить одно значение, являющееся конечным результатом работы функции.

Например, рассмотрим рекурсивную схему вычисления факториала:

$$n! = \begin{cases} 1 & n = 0, \\ n \cdot (n-1)! & n > 0. \end{cases}$$

Данная функция имеет уже в своем определении две ветви: нерекурсивную (первая строка, при n=0) и рекурсивною (вторая строка, при n>0).

Пример 1.2 Пример рекурсивной функции вычисляющей факториал

```
function fact (n: integer): integer;
begin
     if (n<= 0) then
       fact := 1
     else
       fact := n*(fact(n-1));
 end;
```

Реализация функции повторяет ее определение: оператор ветвления имеет нерекурсивный возврат результата (в then блоке), и возврат результата с вызовом самой функции с другим аргументом (в else блоке).

В качестве другого примера, рассмотрим рекурсивную схему вычисления чисел Фибоначчи:

$$F_n = F_{n-1} + F_{n-2},$$
$$F_0 = 0,\ F_1 = 1$$

Пример 1.3 Пример рекурсивной функции вычисляющей факториал

```
function Fib (n: integer): integer;
begin
     if (n = 0) then
       Fib := 0
     else if (n = 1) then
       Fib := 1
     else
       Fib := Fib(n-1) + Fib (n-2);
 end;
```

Данная функция содержит две нерекурсивные ветки (возврат 0 и возврат 1) и одну рекурсивную (возврат Fib(n-1) +Fib(n-2)).

Следует обратить внимание на то, что значение функции зависит только от входных параметров, что делает ее свободной от «побочных эффектов» (side effects).

3 Вопросы для самопроверки

1) Что такое рекурсия?
2) Какие достоинства и недостатки имеются у рекурсивной обработки данных?
3) Опишите принципы работы рекурсивных функций?
4) Почему в рекурсивных функциях необходимо иметь хотя бы одну не рекурсивную ветку?

5 Домашнее задание

5.1 Рекурсивные функции

Создать рекурсивную функцию, реализующую один из следующих алгоритмов обработки последовательности чисел.

1) Последовательность Фибоначчи
2) Факториал (произведение целых чисел от 1 до N)
3) Произведение нечетных чисел от 1 до N
4) Произведение четных чисел от 2 до N
5) Сумма целых чисел от 1 до N
6) Сумма нечетных чисел от 1 до N
7) Сумма четных чисел от 2 до N
8) Степень числа x

5.2 Использование рекурсии при решении задач

5.2.1 Алгоритм Евклида

Найти НОД методом Евклида (алгоритм Евклида). Использовать рекурсивную процедуру.

5.2.2 Острова

Каждый элемент квадратной матрицы размеренности N x N равен нулю, либо единице. Найдите количество «островов», образованных единицами. Под «островом» понимается группа единиц (либо одна единица), со всех сторон окруженная нулями (или краями матрицы). Единицы относятся к одному «острову», если из одной из них можно перейти к другой «наступая» на единицы, расположенные в соседних клетках. Соседними являются клетки, граничащие по горизонтали или вертикали