<p>Согласно математическому определению списка, это конечная последовательность элементов (возможно, пустая). В непустой последовательности можно выделить первый элемент – «голову» последовательности – и оставшуюся часть – «хвост». Заметим, что «хвост» в свою очередь также является последовательностью. Таким образом, получаем рекурсивное определение последовательности: </p>

<pre>

<последовательность>::= <голова> <хвост> | <пусто>

<голова>::= <элемент>

<хвост>::= <последовательность>

</pre>

<p>Опираясь на это определение, можно задать рекурсивную процедуру (функцию) поэлементной обработки последовательности. Пусть, например, требуется напечатать список символов L (тип элементов – char). Процедуру печати можно сформулировать так: если список пуст – ничего не делать (так как нечего печатать), иначе напечатать «голову», а за ней напечатать «хвост». Для печати «хвоста» используем эту же процедуру (рекурсивное применение). Ниже приводятся рекурсивные процедуры print1 и print2 для печати списков соответственно с заглавным звеном и без заглавного звена. </p>

<pre>

procedure print1(L: list); {печать списка с заглавным звеном}

begin

if L <> nil then begin

write(L^.elem,' '); {печать «головы»}

print1(L^.next) {печать «хвоста»}

end;

end;

procedure print2(L: list); {печать списка без заглавного звена}

begin

if L^.next <> nil then begin

write(L^.next^.elem,' ');

print2(L^.next)

end;

end;

</pre>

<p>Нерекурсивный (итерационный) вид, например, процедуры print1, имеет вид:</p>

<pre>

procedure print1(L: list);

var p: list;

begin

p:= L;

while p <> nil do begin

write(p^.elem,' ');

p:= p^.next

end;

end;

</pre>

<p>Как видно, при таком подходе требуется дополнительная локальная переменная.</p>

<p>Рассмотрим ещё одну задачу: «Не используя операторов цикла и перехода, описать функцию sum(L), вычисляющую сумму элементов списка целых чисел L. Считать, что сумма пустого списка равна нулю и что список представлен цепочкой звеньев без заглавного звена». </p>

<p>Поскольку циклы запрещены условием задачи, для прохода по списку воспользуемся рекурсией (как в реализации процедуры print1). Дадим рекурсивное определение суммы списка. Сумма пустого списка равна нулю, сумма непустого списка есть результат сложения «головы» с суммой «хвоста»: </p>

<pre>

sum(L) = 0, если L пуст;

sum(L) = L^.elem + sum(L^.next), иначе.

</pre>

<p>Теперь запишем определение функции на Паскале. </p>

<pre>

function sum(L: list): integer;

begin

if L = nil then sum:= 0

else sum:= L^.elem + sum(L^.next)

end;

</pre>

<p>Параметр L передаётся по значению. Поэтому при каждом вызове этой функции создаётся локальная (по отношению к данному вызову) переменная L, и ей присваивается значение фактического параметра. При выходе из функции локальная переменная исчезает, и значение функции возвращается в точку обращения к ней. При рекурсивном обращении могут одновременно существовать несколько (различных!) локальных переменных с именем L (по одной для каждого вызова). Рисунок ниже иллюстрирует выполнение оператора write(sum(Z)) основной программы, где Z представляет список <5,-3>.</p>

