Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных технология и систем»

Лабораторная работа №6

**«ОБРАБОТКА СПИСКОВ И РЕКУРСИЯ В ЯЗЫКЕ PROLOG»**

по дисциплине:

«Функциональное и логическое программирование»

**Отчёт**

Принял преподаватель:

\_\_\_\_\_\_\_\_\_\_ / Михайлов Д.В

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_\_ / Лехновский А. Д.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Великий Новгород**

**2021**

1. **Цель и задачи данной лабораторной работы**

Целью лабораторной работы является изучение приемов работы со списками в Прологе, а также более детальное изучение рекурсивного программирования применительно к обработке списков.

1. **Задания на лабораторную работу**

**Задача №1:**

Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится методом, предложенным Дональдом Кнутом.

**Задача №2:**

Написать программу сортировки списка методом пузырька.

**Задача №3:**

Написать программу решения головоломки «Пирамида из домино». Требуется расположить комплект домино в виде пирамиды, соблюдая следующие условия:

* В каждой строчке сумма очков на косточках должна быть точным квадратом.
* В строчках косточки укладываются согласно правилам игры в домино: 0 к 0, 1 к 1 и т. д.

**Задача №4:**

Написать программу вставки в список нового элемента на все места, где i задается в качестве аргумента, а n = 1, 2, 3, . . .

1. **Решение поставленных задач**

Для выполнения поставленных задач был использован Visual Prolog 5.2.

**Задача №1:**

Идея сортировки методом Шелла состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

Каждый проход в алгоритме характеризуется смещением  , таким, что сортируются элементы отстающие друг от друга на  позиций. Шелл предлагал использовать . Возможны и другие смещения, но  всегда.

* **Шаг 0.** .
* **Шаг 1.** Разобьем массив на списки элементов, отстающих друг от друга на . Таких списков будет .
* **Шаг 2.** Отсортируем элементы каждого списка сортировкой вставками.
* **Шаг 3.** Объединим списки обратно в массив. Уменьшим . Если  неотрицательно — вернемся к шагу 1

Количество шагов сортировки (разделение на группы) и шаги (расстояния) в данной программе вычисляются методом, предложенным Дональдом Кнутом.

Количество шагов сортировки вычисляется по формуле:

где – количество элементов в списке.

Смещение или шаг вычисляется по формуле:

Первоначально расстояние у нас равно . На первом шаге каждая группа включает в себя элементы, расположенные друг от друга на расстоянии . Эти элементы сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние сокращается. Постепенно расстояние между элементами уменьшается, и на проход по списку происходит в последний раз.

Правила sort\_shell запускают нашу сортировку списка:

* В случаях, когда список пустой или список содержит 1 элемент, нам не требуется высчитывать шаг и количество шагов, потому что такой список уже является отсортированным.
* Также если у нас список состоит из двух элементов, то происходит сравнение двух элементов списка, после либо производиться обмен, либо нет, затем на выходе получаем отсортированный список.
* В остальных случаях, запускается сортировка Шелла, где для вычисления количества шагов сортировки отвечает правило count\_steps, а также используется правило list\_steps для вычисления последовательности значений шагов сортировки.

Код программы для решения задачи приведен в *Приложении 1*.

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| [] | [] |
| [22] | [22] |
| [8,22] | [8,22] |
| [13,1] | [1,13] |
| [67, 1, 8, 3, -1, 5] | [-1,1,3,5,8,67] |
| [1,2,5, 22, 7, 1, 0, -1, 2, 2 , 5, 6] | [-1,0,1,1,2,2,2,5,5,6,7,22] |

*Таблица №1. Тестовые наборы данных.*

**Задача №2:**

**Сортировка простыми обменами**, **сортировка пузырьком** (англ. *bubble sort*) — простой в реализации и малоэффективный алгоритм сортировки. Алгоритм состоит в повторяющихся проходах по сортируемому списку. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по списку как минимум один элемент встает на свое место, поэтому необходимо совершить не более проходов, чем размер списка, чтобы отсортировать список.

Правило bubble\_sort запускает сортировку списка методом пузырька:

Если список пустой, то возвращает пустой список; если список состоит из одного элемента, то возвращается этот элемент.

В противном случае, начинается первый проход по списку, где хвост исходного списка после каждого прохода будет уменьшаться на единицу, также будет отдельно храниться отсортированный за проход список, который не включает максимальный элемент за данный проход, и сам максимальный элемент за проход. В результате первого прохода на последнее место "всплывёт" максимальный элемент. Теперь снова обходим неотсортированную часть списка (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть списка, убирая найденные максимумы в конец.

Код программы для решения задачи приведен в *Приложении 2*.

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| [] | [] |
| [100] | [100] |
| [100, 55, 27] | [27,55,100] |
| [34, 35] | [34,35] |
| [55, 54] | [54,55] |
| [3,1,7,2,9,0,11,5,23,6] | [0,1,2,3,5,6,7,9,11,23] |

*Таблица №2. Тестовые наборы данных.*

Эффективность сортировки пузырьком**:.**

Эффективность сортировки Шелла, которая в качестве вычисления последовательности шагов использует метод, предложенный Дональдом Кнутом: **.**

В сравнение этих двух алгоритмов по вычислительной сложности, более эффективным оказался метод сортировки Шелла, которая в качестве вычисления последовательности шагов использует метод, предложенный Дональдом Кнутом. Метод пузырька гораздо менее эффективен других алгоритмов, однако он имеет простую и понятную реализацию. Кроме того, пузырьковая сортировка может использоваться для работы с небольшими массивами данных.

**Задача №3:**

Домино — настольная игра, в процессе которой выстраивается цепь костяшек, соприкасающихся половинками с одинаковым количеством точек, обозначающим число очков. Количество костяшек в классическом домино равно 28 костяшек, сумма которых равна 168.

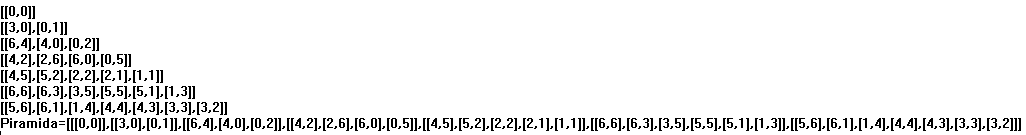
Используя условия:

* В каждой строчке сумма очков на косточках должна быть точным квадратом.
* В строчках косточки укладываются согласно правилам игры в домино: 0 к 0, 1 к 1 и т. д.

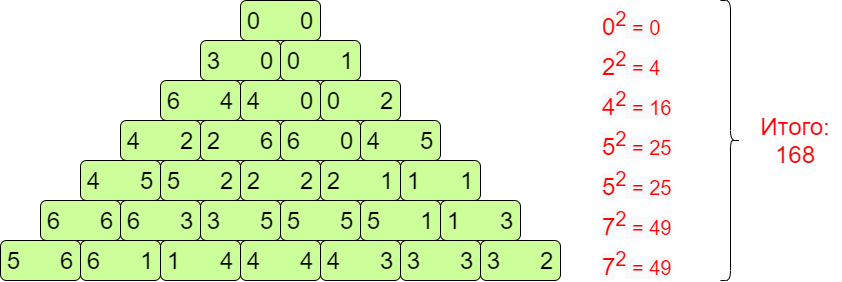
Идея программы заключается в том, чтобы начать строить пирамиду с вершины и делать это построчно, так чтобы в каждой строке срабатывали поставленные в этой задачи ограничения. Костяшки из домино будет представлять в виде списков, состоящих из двух элементов (значения левого и правого конца). Само домино представим списком, где его элементами являются костяшки, то есть подсписки. У списка пирамиды будем представлять списком строк пирамиды, где сама строка есть список, задействованных в ней костяшек.

Правила mem будут выбирать одну костяшку из доступных, а также возвращают список оставшихся (незадействованных в пирамиде) костяшек и ещё рассматривают, если у костяшки разные концы, то они также возвращают и вариант, когда данная костяшка перевернута. А правила getStr будут строить строку пирамиды согласно, наложенным ограничениям.

Код программы для решения задачи приведен в *Приложении 3*.

Пример работы программы:

*Рисунок №1. Результат работы программы для задачи 3.*



*Рисунок №2. Пирамида из домино.*

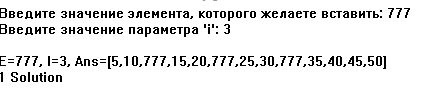
Итог, были задействованы все 28 костяшек для построения пирамиды из домино, а значит сумма костяшек равна 168. Плюс, для проверки, что наше решение выполняет поставленные условия, давайте посчитаем сумма костяшек в каждой строке пирамиды (см. рис. №2), а второе условие и так видно, что выполняется.

**Задача №4:**

За вставку в список одно определённого элемента, который будет находиться на местах, где i задается в качестве аргумента, а в n = 1, 2, 3 … отвечают правила ins:

* если на вход подается пустой список и аргумент i =1, то результатом будет непустой список;
* если на вход подается пустой список и i >1, то результатом будет уже пустой список;
* если на вход подается непустой список, аргумент i, Numbers - список значений n,который генерируется в зависимости от длины подаваемого списка, и , то производится вставка на указанные места . На места, где условие не выполняется вставка не производится.

Пример работы программы для списка [5,10,15,20,25,30,35,40,45,50], когда **i** и **E**(Вставка) вводятся с клавиатуры:



*Рисунок №3. Результат работы программы для Задачи №4.*

Так как длина исходного списка равна 10, тогда список Numbers тоже будет состоять из 10 элементов и получиться следующим: [1,2,3,4,5,6,7,8,9,10]. (I \* N) < длина исходного списка.

Код программы для решения задачи приведен в *Приложении 4*.

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| E=100, I =1, List=[] | [100] |
| E=100, I =2, List=[] | [] |
| E=100, I =2, List= [11,22,33,44,55,66,77,88,99] | [11,100,22,100,33,100,44,100,55,66,77,88,99] |
| E=-1, I =3, List= [0,0,0,0,0,0,0,0,1] | [0,0,-1,0,0,-1,0,0,-1,0,0,1] |
| E=22, I =2, List= [0,0,0,0,0,0,0,0,0] | [0,22,0,22,0,22,0,22,0,0,0,0,0] |

*Таблица №3. Тестовые наборы данных.*

1. **Вывод**

В данной лабораторной работе, мною были изучены приемы работы со списками в Прологе, а также реализованы сортировка Шелла, сортировка пузырьком, вставка элементов в список по заданным условиям и решена логическая головоломка «пирамида из домино».

**Приложение 1**

Код программы:

DOMAINS

ilst = integer\* %список, содержащий переменные типа int

PREDICATES

nondeterm length\_lst(ilst, integer)

nondeterm distance\_lst(ilst, integer, integer)

nondeterm power(integer, integer, integer)

nondeterm insert\_in\_lst(ilst, integer, integer, ilst)

nondeterm remove\_from\_lst(ilst, integer, ilst)

nondeterm get\_element(ilst, integer, integer)

nondeterm swap\_elements(ilst, integer, integer, ilst)

nondeterm list\_steps(integer, ilst)

nondeterm list\_steps\_temp(integer, ilst)

nondeterm count\_steps(integer, integer, integer)

nondeterm value\_step(integer,integer)

nondeterm sort\_shell(ilst,ilst)

nondeterm sort\_shell\_steps(ilst,ilst,ilst)

nondeterm sort\_shell\_step(ilst,integer,ilst)

nondeterm sort\_shell\_step\_value(ilst,integer,integer,integer,ilst)

CLAUSES

%Частные случаи

%Если список пустой

sort\_shell([],[]):-!.

%Если список из одного элемента

sort\_shell(List, List):-

length\_lst(List, Length),

Length = 1,!.

%Если список состоит из двух элементов, где Element\_1 < Element\_2

sort\_shell([Element\_1, Element\_2|Tail],[Element\_1, Element\_2]):-

length\_lst([Element\_1, Element\_2|Tail],Length),

Length = 2,

Element\_1 < Element\_2,!.

%Если список состоит из двух элементов, где Element\_1 > Element\_2

sort\_shell([Element\_1, Element\_2|Tail],[Element\_2,Element\_1]):-

length\_lst([Element\_1, Element\_2|Tail],Length),

Length = 2,

Element\_1 > Element\_2,!.

%Сортировка Шелла

%С использованием метода для вычисления последовательности шагов сортировки,предложенным Дональдом Кнутом.

sort\_shell(List,SortedList):-

length\_lst(List, Length),

list\_steps(Length, Steps),

sort\_shell\_steps(List, Steps,SortedList),!.

%Дальше пойдут шаги сортировки

sort\_shell\_steps(SortedList,[],SortedList).

sort\_shell\_steps(List,[Head|Tail],SortedList):-

sort\_shell\_step(List,Head,TempList),

sort\_shell\_steps(TempList, Tail, SortedList).

sort\_shell\_step(List, ValueStep, Res):-

sort\_shell\_step\_value(List,ValueStep,0,ValueStep,Res).

sort\_shell\_step\_value(Res, ValueStep, Numb\_1, \_, Res):-

length\_lst(Res, Length),

Max = Numb\_1 + ValueStep,

Max >= Length.

sort\_shell\_step\_value(List, ValueStep, Numb\_1, Numb\_2, Res):-

length\_lst(List, Length),

Numb\_2 >= Length,

Numb\_3 = Numb\_1 + 1,

Numb\_4 = Numb\_3 + 1,

sort\_shell\_step\_value(List, ValueStep, Numb\_3, Numb\_4, Res).

sort\_shell\_step\_value(List, ValueStep, Numb\_1, Numb\_2, Res):-

get\_element(List,Numb\_1,Element\_1),

get\_element(List,Numb\_2,Element\_2),

Element\_1 > Element\_2,

swap\_elements(List,Numb\_1,Numb\_2,NewList),

Numb\_3 = Numb\_2 + ValueStep,

sort\_shell\_step\_value(NewList,ValueStep, Numb\_1, Numb\_3, Res).

sort\_shell\_step\_value(List, ValueStep, Numb\_1, Numb\_2, Res):-

get\_element(List,Numb\_1,Element\_1),

get\_element(List,Numb\_2,Element\_2),

Element\_1 <= Element\_2,

Numb\_3 = Numb\_2 + ValueStep,

sort\_shell\_step\_value(List,ValueStep, Numb\_1, Numb\_3, Res).

%Перестановка элементов в списке

swap\_elements(List,Numb\_1,Numb\_2,NewList):-

get\_element(List,Numb\_1, Element\_1),

get\_element(List,Numb\_2, Element\_2),

remove\_from\_lst(List,Numb\_1,List\_1),

insert\_in\_lst(List\_1,Element\_2,Numb\_1,List\_2),

remove\_from\_lst(List\_2,Numb\_2,List\_3),

insert\_in\_lst(List\_3,Element\_1,Numb\_2,NewList).

%Правило для вычисления степени числа

power(\_,0, 1):- !.

power(A,B, Pow):-

B > 0, 0 <> B mod 2, !,

TailB = B - 1,

power(A, TailB, TailPow),

Pow = TailPow \* A.

power(A, B, Pow):-

B > 0, 0 = B mod 2, !,

TailB = B / 2,

power(A, TailB, TailPow),

Pow = TailPow \* TailPow.

power(A, B, Pow):-

B1 = B \* (-1),

power(A, B1, Divider),

Pow = 1/Divider.

/\*

Первый агрумент - исходный список

Второй аргумент - элемент, который хотим вставить

Третий агрумент - номер или место в списке, на которое хотим поставить данный элемент

Чётвертый агрумент - результат

\*/

%Правило для вставки элемента в список, индексация с нуля

insert\_in\_lst([],Element,0,[Element]).

insert\_in\_lst([Head|Tail],Element,0,[Element|[Head|Tail]]).

insert\_in\_lst([Head|Tail],Element,Numb,[Head|Tail\_1]):-

Numb\_1 = Numb - 1,

insert\_in\_lst(Tail,Element,Numb\_1,Tail\_1).

%Удаление элемента из списка, индексация с нуля

remove\_from\_lst([\_|Tail],0,Tail).

remove\_from\_lst([Head|Tail],Numb,[Head|Tail\_1]):-

Numb\_1 = Numb - 1,

remove\_from\_lst(Tail,Numb\_1,Tail\_1).

%Получить значение элемента по индексу, индексация с нуля

get\_element([Head|\_],0,Head).

get\_element([\_|[Head\_1|Tail]],Numb,Element):-

Numb\_1 = Numb - 1,

get\_element([Head\_1|Tail],Numb\_1,Element).

%Правила для вычисления длины списка

length\_lst(List, Length):-

distance\_lst(List, Length, 0).

distance\_lst([], Length, Length).

distance\_lst([\_|Tail], Length, Length\_1):-

Length\_2 = Length\_1 + 1,

distance\_lst(Tail, Length, Length\_2).

%Правила для получения последовательности шагов

list\_steps(Length, Steps) :-

count\_steps(Length, 2, CountSteps),

list\_steps\_temp(CountSteps, Steps).

list\_steps\_temp(CountSteps, Steps) :-

CountSteps > -1, !,

value\_step(CountSteps, Value),

NewCountSteps = CountSteps - 1,

list\_steps\_temp(NewCountSteps, StepsTail),

Steps = [Value|StepsTail];

Steps = [].

%Правило получения количества шагов сортировки

count\_steps(Length, Start, Steps) :-

Steps = Start - 2,

power(3,Start, Power),

Length < Power;

NextStart = Start + 1,

count\_steps(Length, NextStart, Steps).

%Получение значения шагов сортировки (смещение)

value\_step(0, 1).

value\_step(Step, Value) :-

Step1 = Step - 1,

value\_step(Step1, Value1),

Value = 3 \* Value1 + 1.

GOAL

sort\_shell([],Ans1),

sort\_shell([22],Ans2),

sort\_shell([8,22],Ans3),

sort\_shell([13,1],Ans4),

sort\_shell([67, 1, 8, 3, -1, 5],Ans5),

sort\_shell([1,2,5, 22, 7, 1, 0, -1, 2, 2 , 5, 6], Ans6).

**Приложение 2**

Код программы:

%Сортировка Пузырьком

DOMAINS

lst = integer\* %lst -это список, состоящий из переменных, имеющих тип integer

PREDICATES

nondeterm bubble\_sort(lst,lst)

nondeterm b\_sort(lst,lst,lst)

nondeterm bubble(integer, lst, lst, integer)

CLAUSES

%bubble\_sort(List,Sorted) - Запускает сортировку пузырьком

bubble\_sort(List,Sorted):- b\_sort(List,[],Sorted).

b\_sort([],Acc,Acc).

/\*

Переменная T - это хвост исходного списка, который с каждым проходом будет уменьшаться на единицу до пустого списка

Переменная NT - это отсортированная за проход список,который не включает максимальный элемент за данный проход

Переменная Max - максимальный элемент за проход

\*/

b\_sort([H|T],Acc,Sorted):- bubble(H,T,NT,Max),b\_sort(NT,[Max|Acc],Sorted).

/\*

В результате первого прохода на последнее место "всплывёт" максимальный элемент.

Теперь снова обходим неотсортированную часть списка (от первого элемента до предпоследнего)

и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте.

Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть списка,

запихивая найденные максимумы в конец

\*/

bubble(X,[],[],X). %Конец рекурсии, находим максимальный элемент X и присваимаем Max = X

bubble(X,[Y|T],[Y|NT],Max):- X>Y, bubble(X,T,NT,Max). %Для случая, если первый элемент больше второго

bubble(X,[Y|T],[X|NT],Max):- X<=Y, bubble(Y,T,NT,Max). % Для случая, если второй элемент больше первого

GOAL

bubble\_sort([],Res1),

bubble\_sort([100],Res2),

bubble\_sort([100, 55, 27],Res3),

bubble\_sort([34, 35],Res4),

bubble\_sort([55, 54],Res5),

bubble\_sort([3,1,7,2,9,0,11,5,23,6],Res6).

**Приложение 3**

Код программы:

DOMAINS

il=integer\* ill=il\* illl=ill\*

CONSTANTS

domino=[[0,0],[0,1],[0,2],[0,3],[0,4],[0,5],[0,6],[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[2,2],[2,3],[2,4],[2,5],[2,6],[3,3],[3,4],[3,5],[3,6],[4,4],[4,5],[4,6],[5,5],[5,6],[6,6]]

PREDICATES

nondeterm piramida(integer,ill,illl)

nondeterm getStr(integer,integer,ill,ill,ill,ill)

nondeterm mem(ill,il,ill)

print\_lst\_col(illl)

CLAUSES

piramida(\_,[],[]):- !.

piramida(A,B,[C|D]):- getStr(A,0,[],C,B,E), F=A+1, piramida(F,E,D).

%Работа со строкой пирамиды

%Первый аргумент - количество элементов, с которыми работаем в данной строке

%Второй аргумет - сумма костяшек домино в данной строке

getStr(0,A,B,B,C,C):- !, D=sqrt(A), D=round(D). %Для соблюдения правила:"В каждой строчке сумма очков на косточках должна быть точным квадратом"

getStr(A,0,[],B,C,D):- mem(C,[X,Y],E), F=X+Y, G=A-1, getStr(G,F,[[X,Y]],B,E,D).

%В строчках косточки укладываются согласно правилам игры в домино: 0 к 0, 1 к 1 и т. д.

getStr(A,B,[[C,D]|E],F,G,H):- mem(G,[I,C],J), S=B+I+C, W=A-1, getStr(W,S,[[I,C],[C,D]|E],F,J,H).

%Правила mem выбирает одну костяшку и возвращает список из оставшихся

mem([A|B],A,B).

mem([[A,B]|C],[B,A],C):- A<>B. %Если у костяшки разные концы, то рассматриваем и второй вариант ([1,0] и [0,1], ...)

mem([A|B],C,[A|D]):- mem(B,C,D). %Рассматриваем и другие костяшки

%Печать списка стобцом

print\_lst\_col([]).

print\_lst\_col([Head|Tail]):-

write(Head),nl,print\_lst\_col(Tail).

GOAL

piramida(1,domino,Piramida),!,print\_lst\_col(Piramida).

**Приложение 4**

Код программы:

DOMAINS

ilist=integer\* %Список, состоящий из переменных типа integer

anslst= ilist\*

PREDICATES

ins(integer,integer,integer,ilist,ilist,ilist).

nondeterm member(integer,ilist).

length\_lst(ilist, integer)

distance\_lst(ilist, integer, integer)

nondeterm get\_Numbers(ilist,integer, ilist)

print\_my(anslst)

CLAUSES

%К - индекс элемента списка,начиная с единицы

%I - аргумент 'i',который мы вводим сами

%E - втавка,то есть элемент,который мы желаем вставить

%Numbers - список значений n,который формируется по длине исходного списка. Numbers=[1,2,3,...]

%L - иходный список, с которым мы будем работать

%Последний аргумент - это результат, то есть получившийся после вставок список

% Для случая, когда мы дошли до места вставки n\*i

ins(K,I,E,\_,L,[E|Tail]):-

get\_Numbers(L,1,Numbers),

member(N,Numbers),

K=N\*I,!,

K1=K+1,

ins(K1,I,E,Numbers,L,Tail).

%Для случая, когда мы не дошли до нужного места, и просто идём дальше по списку

ins(K,I,E,Numbers,[H|Tail],[H|Tail1]):- K1=K+1,ins(K1,I,E,Numbers,Tail,Tail1).

ins(\_,\_,\_,\_,[],[]).

%Получение списка значений N (в зависимости от длины исходного списка)

get\_Numbers([],\_,[1]).

get\_Numbers(List,N,ListN):-

length\_lst(List, Length),

Length >= N, !,

NewN = (N + 1),

length\_lst(List, Length),

get\_Numbers(List,NewN,ListNTail),

ListN = [N|ListNTail];

ListN = [].

%Правила member вытаскивают значения n из списка Numbers

member(H,[H|\_]).

member(H,[\_|Tail]):- member(H,Tail).

%Правила для вычисления длины списка

length\_lst(List, Length):-

distance\_lst(List,Length, 0).

distance\_lst([],Length,Length).

distance\_lst([\_|Tail],Length,Length\_1):-

Length\_2 = Length\_1 + 1,

distance\_lst(Tail,Length,Length\_2).

%Функция для печати нескольких результатов

print\_my([]).

print\_my([X|Y]) :-

write("Результат: "),write(X), nl,

print\_my(Y).

GOAL

% readint - встроенная функиця для считывания с клавиатуры целого числа

write("Введите значение элемента, которого желаете вставить: "),readint(E),

write("Введите значение параметра 'i': "),readint(I),

ins(1,I,E,[],[5,10,15,20,25,30,35,40,45,50],Ans),nl, %Для случая ввода с клавиатуры

%Остальные примеры

ins(1,1,100,[],[],Ans1),

ins(1,2,100,[],[],Ans2),

ins(1,2,100,[],[11,22,33,44,55,66,77,88,99],Ans3),

ins(1,3,-1,[],[0,0,0,0,0,0,0,0,1],Ans4),

ins(1,2,22,[],[0,0,0,0,0,0,0,0,0],Ans5),nl,

print\_my([Ans,Ans1,Ans2,Ans3,Ans4,Ans5]),nl.