**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Базы знаний и экспертные системы»**

**Тема: Рекурсивные структуры данных (списки)**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Шевелева А.М. |
| Преподаватель |  | Сучков А.И. |

Санкт-Петербург

2019

Цель работы.

Изучение и исследование рекурсивных структур данных в языке Visual Prolog на примере списков.

Основные теоретические положения

Пролог позволяет определить и использовать рекурсивные типы данных. Примерами рекурсивных типов данных служат списки и деревья. Список – это объект данных, содержащий конечное число других объектов (элементов списка). Список, содержащий числа 1, 2 и 3, записывается следующим образом: [1, 2, 3].

Для объявления списка используется следующее описание домена, представленное на рис. 1.

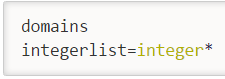


Рисунок 1 − Объявление списка

Список является рекурсивным составным объектом, он состоит из двух частей:

1. головы списка – первого элемента списка;
2. хвоста списка – списка, включающего все последующие элементы.

Пусть имеется список [1|[2, 3]]. Тогда головой списка будет элемент 1, а хвостом [2, 3].

Так как список имеет рекурсивную составную структуру, для работы со списками используется рекурсия.

Постановка задачи.

Реализовать на языке Visual Prolog программу, выполняющую заданные операции над списками в соответствии с индивидуальным вариантом задания. Порядок выполнения работы следующий:

1. Написать на языке Visual Prolog программу, реализующую заданные операции над списками в соответствии с 18 вариантом задания.
2. Произвести отладку программы в системе Visual Prolog для запросов на решение прямой и обратной задачи и задачи на перебор вариантов.
3. Построить трассу программы при выполнении каждого запроса.

Выполнение работы.

Написана программа на языке Visual Prolog циклического сдвига списка вправо на заданное число элементов. Полный код программы смотреть в приложении А.

Чтобы запустить приложение, нужно нажать кнопку «R» на панели инструментов. В окне приложения «GUI\_lab3\_1» выбрать вкладку «List» и в выпадающем списке вкладку «Shift». После этого появится окно программы.

В программе вводится список чисел, разделенных пробелами, в первом текстовом поле и число, на которое будет проводиться сдвиг по списку вправо, во втором текстовом поле. Далее нажимается кнопка «Результат». В третьем текстовом поле появляется результат сдвига списка вправо. Внешний вид программы показан на рис. 2.

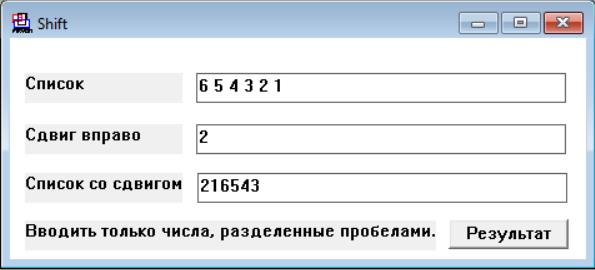


Рисунок 2 − Окно программы сдвига списка

Проверено решение обратной задачи и задачи на перебор вариантов на консольном приложении данной программы. Полный код приложения смотреть в приложении Б. Обратная задача - известен список со сдвигом и сдвиг, нужно получить изначальный список. Запустить приложение не удалось. Результат представлен на рис. 3.

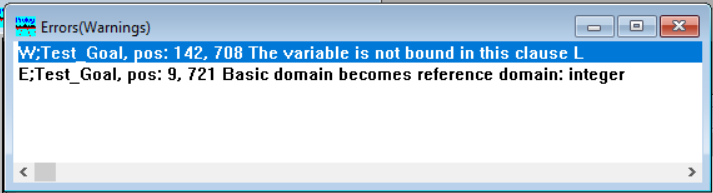


Рисунок 3 − Запуск обратной программы сдвига списка

Задача перебора запускается не при всех неизвестных. Если среди неизвестных присутствует начальный список, то программа не запускается. Ошибка при запуске аналогична ошибке при запуске обратной программы. Если неизвестен только сдвиг, программа корректно выдает сдвиг, при котором мог получиться результирующий список (см. рис. 4). Если неизвестен сдвиг и результирующий список, программа выдает сдвиг, при котором результирующий список совпадает с начальным списком (см. рис. 5).

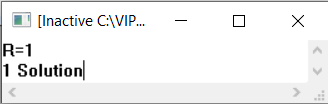


Рисунок 4 − Запуск задачи на перебор (при неизвестном сдвиге)

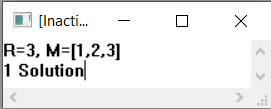


Рисунок 5 − Запуск задачи на перебор (при неизвестном сдвиге и результирующем списке)

Трасса выполнения программы представлена в приложении В.

Написана программа на языке Visual Prolog, подсчитывающая количество элементов списка без какого-либо указываемого элемента. Полный код программы смотреть в приложении Г.

Чтобы запустить приложение, нужно нажать кнопку «R» на панели инструментов. В окне приложения «GUI\_lab3\_1» выбрать вкладку «List» и в выпадающем списке вкладку «Count». После этого появится окно программы.

В программе вводится список символов, не разделенных пробелами, в первом текстовом поле и символ, без которого будет считаться длина списка, во втором текстовом поле. Далее нажимается кнопка «Результат». В третьем текстовом поле появляется длина списка. Внешний вид программы показан на рис. 6.

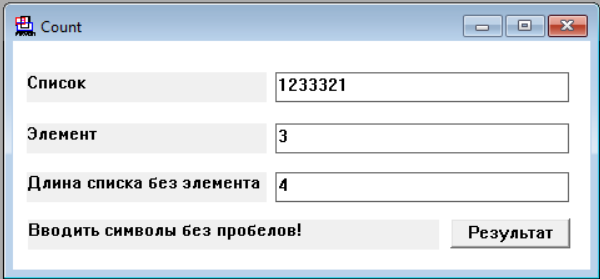


Рисунок 6 − Окно программы поиска длины списка без определенного элемента

Проверено решение обратной задачи и задачи на перебор вариантов на консольном приложении данной программы. Полный код приложения смотреть в приложении Д. Обратная задача - известна длина списка без определенного элемента и элемент, нужно получить изначальный список. Запустить приложение не удалось. Результат представлен на рис. 7.



Рисунок 7 − Запуск обратной программы поиска длины списка

Задача на перебор также не запускается при любых комбинациях неизвестных. Результат запуска представлен на рис. 8.

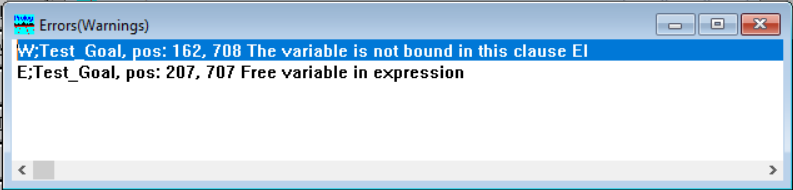


Рисунок 8 − Запуск задачи на перебор

Трасса выполнения программы представлена в приложении Е.

Написана программа на языке Visual Prolog, которая выводит множество элементов списка Z − разность множеств элементов списков X и Y. Полный код программы смотреть в приложении Ж.

Чтобы запустить приложение, нужно нажать кнопку «R» на панели инструментов. В окне приложения «GUI\_lab3\_1» выбрать вкладку «List» и в выпадающем списке вкладку «Sub». После этого появится окно программы.

В программе вводятся в первое и второе текстовое поле соответственно списки X и Y. Далее нажимается кнопка «Результат». В третьем текстовом поле появляется разность списков X и Y. Внешний вид программы показан на рис. 9.

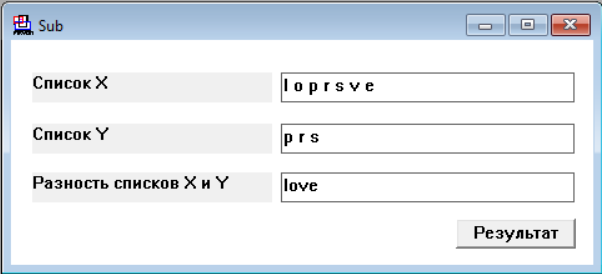


Рисунок 9 − Окно программы поиска разности списков

Проверено решение обратной задачи и задачи на перебор вариантов на консольном приложении данной программы. Полный код приложения смотреть в приложении И. Обратная задача - известны второй список и результирующий список, нужно получить первый список. Запустить приложение не удалось. Результат представлен на рис. 10.

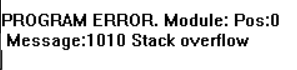


Рисунок 10 − Запуск обратной программы поиска длины списка

Задача на перебор также не запускается при любых комбинациях неизвестных. Результат запуска представлен на рис. 11.

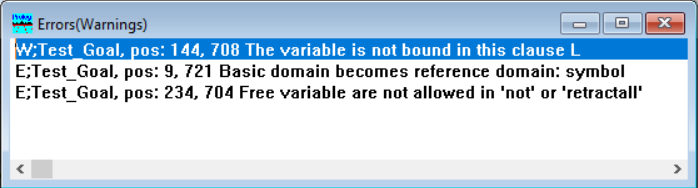


Рисунок 11 − Запуск задачи на перебор

Трасса выполнения программы представлена в приложении К.

Выводы.

В программе Visual Prolog была написана программа, реализующая 3 задачи работы со списками:

1. Сдвиг списка вправо на заданное количество элементов;
2. Поиск длины списка, без заданного элемента;
3. Определение списка, являющегося разностью двух заданных списков.

Для программы было разработано GUI-приложение.

Для каждой задачи было создано консольное приложение, в котором проверялась работа обратной задачи и работа задачи по перебору. Также для каждой задачи была создана трасса выполнения программы.

**ПРИЛОЖЕНИЕ А**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ СПИСОК СО СДВИГОМ ВПРАВО, GUI ПРИЛОЖЕНИЕ**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) My Company

Project: GUI\_LAB3

FileName: GUI\_SHIFT.PRO

Purpose: No description

Written by: Visual Prolog

Comments:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

include "gui\_lab3.inc"

include "gui\_lab3.con"

include "hlptopic.con"

%BEGIN\_WIN Shift

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Creation and event handling for window: Shift

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

domains

strlist = string\*

list = integer\*

int = integer

constants

%BEGIN Shift, CreateParms, 22:58:04-22.10.2019, Code automatically updated!

win\_shift\_WinType = w\_TopLevel

win\_shift\_Flags = [wsf\_SizeBorder,wsf\_TitleBar,wsf\_Maximize,wsf\_Minimize,wsf\_Close,wsf\_ClipSiblings,wsf\_ClipChildren]

win\_shift\_RCT = rct(100,80,505,259)

win\_shift\_Menu = no\_menu

win\_shift\_Title = "Shift"

win\_shift\_Help = idh\_contents

%END Shift, CreateParms

predicates

append(list, list, list)

length(list,int)

shift\_right(list, int, list)

string\_namelist(string,strlist)

namelist\_string(strlist,string)

convertStrListToIntList(strlist,list)

csltil(list,list,string)

int2str(integer,string)

task(list,strlist)

win\_shift\_eh : EHANDLER

clauses

append([], L, L).

append([X|M], L, [X|N]) :- append(M, L, N).

length([],0).

length( [ \_ | Tail ],N) :- length(Tail,N1), N = N1 + 1.

shift\_right(List, N, Slided):-

append(Left, Right, List),

length(Right, N),

append(Right, Left, Slided), !.

string\_namelist(S,[H|T]):- fronttoken(S,H,S1),!, string\_namelist(S1,T).

string\_namelist(\_,[]).

namelist\_string([], ""):-!.

namelist\_string([Head|ListTail], String):- namelist\_string(ListTail, StringTail), fronttoken(String, Head, StringTail).

convertStrListToIntList([],[]):-!.

convertStrListToIntList([HS|TS],LI):- convertStrListToIntList(TS,TI), csltil(TI,LI,HS).

csltil(TL,[HI|TL],HS):- str\_int(HS,HI),!.

csltil(TL,TL,\_):-!.

int2str(N,S) :- N<=9, M=48+N, char\_int(Q,M), frontchar(S,Q,""),!.

int2str(N,S) :- M=(N mod 10), MM=48+M, char\_int(Q,MM), frontchar(Z,Q,""), D=(N div 10), int2str(D,SS), concat(SS,Z,S).

task([],[]).

task([H|T],[SH|ST]) :- int2str(H,SH), task(T,ST).

win\_shift\_Create(\_Parent):-

win\_Create(win\_shift\_WinType,win\_shift\_RCT,win\_shift\_Title,

win\_shift\_Menu,\_Parent,win\_shift\_Flags,win\_shift\_eh,0).

%BEGIN Shift, e\_Create

win\_shift\_eh(\_Win,e\_Create(\_),0):-!,

%BEGIN Shift, InitControls, 22:58:04-22.10.2019, Code automatically updated!

win\_CreateControl(wc\_PushButton,rct(294,135,390,159),"Результат",\_Win,[wsf\_Group,wsf\_TabStop],idc\_push\_button),

win\_CreateControl(wc\_Text,rct(13,21,139,45),"Список",\_Win,[wsf\_AlignLeft],idct\_список),

win\_CreateControl(wc\_Text,rct(13,58,139,82),"Сдвиг вправо",\_Win,[wsf\_AlignLeft],idct\_сдвиг\_вправо),

win\_CreateControl(wc\_Text,rct(13,96,139,120),"Список со сдвигом",\_Win,[wsf\_AlignLeft],idct\_список\_со\_сдвигом),

win\_CreateControl(wc\_Edit,rct(160,21,390,45),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit1),

win\_CreateControl(wc\_Edit,rct(160,58,390,82),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit2),

win\_CreateControl(wc\_Edit,rct(160,96,390,120),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit3),

%END Shift, InitControls

%BEGIN Shift, ToolbarCreate, 22:58:04-22.10.2019, Code automatically updated!

%END Shift, ToolbarCreate

!.

%END Shift, e\_Create

%MARK Shift, new events

%BEGIN Shift, e\_Size

win\_shift\_eh(\_Win,e\_Size(\_Width,\_Height),0):-!,

ifdef use\_tbar

toolbar\_Resize(\_Win),

enddef

!.

%END Shift, e\_Size

%BEGIN Shift, e\_Menu, Parent window

win\_shift\_eh(Win,e\_Menu(ID,CAS),0):-!,

PARENT = win\_GetParent(Win),

win\_SendEvent(PARENT,e\_Menu(ID,CAS)),

!.

win\_shift\_eh(\_Win,e\_Control(idc\_push\_button,\_CtrlType,\_CtrlWin,\_CtlInfo),0):-

!, Ctl1=win\_GetCtlHandle(\_Win,idc\_edit1),

String1 = win\_GetText(Ctl1),

string\_namelist(String1, Str),

convertStrListToIntList(Str,X),

Ctl2=win\_GetCtlHandle(\_Win,idc\_edit2),

String2 = win\_GetText(Ctl2),

str\_int(String2,Y),

shift\_right(X, Y, Z1),

task(Z1,Z),

namelist\_string(Z,String),

Ctl3=win\_GetCtlHandle(\_Win,idc\_edit3),

win\_SetText(Ctl3,String),

!.

%END Shift, e\_Menu, Parent window

%END\_WIN Shift

**ПРИЛОЖЕНИЕ Б**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ СПИСОК СО СДВИГОМ ВПРАВО, КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ**

domains

list = integer\*

int = integer

predicates

append(list, list, list)

length(list,int)

shift\_right(list, int, list)

clauses

append([], L, L).

append([X|M], L, [X|N]) :- append(M, L, N).

length([],0).

length( [ \_ | Tail ],N) :- length(Tail,N1), N = N1 + 1.

shift\_right(List, N, Slided):-

append(Left, Right, List),

length(Right, N),

append(Right, Left, Slided), !.

goal

shift\_right([1,2,3,4], 2, L).

**ПРИЛОЖЕНИЕ В**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ПРОГРАММЫ СДВИГА СПИСКА**

Трасса выполнения запроса сдвига на один символ вправо списка [1,2] представлена в табл. 1.

Таблица 1 − Трасса выполнения запроса сдвига

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Переменная | Стек вызовов | Пояснение работы |
| 1 | R = \_ | 1 \_PROLOG\_Goal() | Просмотр цели prolog-программы |
| 2 | List = [1,2]  N = 1  Slided = \_  Left = \_  Right = \_ | 1 shift\_right([1,2], 1, \_)  2 \_PROLOG\_Goal() | Добавление цели в Call Stack. |
| 3 | List = [1,2] | 1 append(\_, \_, [1,2])  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Просмотр правила объединения строк. Добавление его в Call Stack. Проверка правила. |
| 4 | List = [1,2]  N = 1  Slided = \_  Left = []  Right = [1,2] | 1 shift\_right([1,2], 1, \_)  2 \_PROLOG\_Goal() | Правило вернуло Right. Убираем его из Call Stack. Переходим к правилу нахождения длины списка. |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | - | 1 length( [1,2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Добавление его в Call Stack. |
| 6 | Tail = \_  N = 1  N1 = \_ | 1 length( [1,2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Вход в рекурсию правила. Поиск длины списка [1,2]. |
| 7 | Tail = \_  N = \_  N1 = \_ | 1 length( [2], \_)  2 length( [1,2], 1)  3 shift\_right([1,2], 1, \_)  4 \_PROLOG\_Goal() | Поиск длины списка [2]. |
| 8 | Tail = \_  N = 1  N1 = 1 | 1 length( [1,2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Удаление из Call Stack последней итерации рекурсии. Результат длины хвоста 1. |
| 9 | X = \_  M = \_  L = \_  N = \_ | 1 append(\_, \_, [1,2])  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Удаление из Call Stack правила поиска длины списка. Переход к правилу объединения. |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | List = [1,2]  N = 1  Slided = \_  Left = [1]  Right = [2] | 1 shift\_right([1,2], 1, \_)  2 \_PROLOG\_Goal() | Переписан один символ в Left из Right. |
| 11 | - | 1 length( [2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Проверка длины списка Right. |
| 12 | Tail = \_  N = 1  N1 = \_ | 1 length( [2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Вход в рекурсию правила. |
| 13 | - | 1 length( [], \_)  2 length( [1,2], 1)  3 shift\_right([1,2], 1, \_)  4 \_PROLOG\_Goal() | Проверка длины пустого списка. |
| 14 | Tail = \_  N = 1  N1 = 0 | 1 length( [2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Проверка окончена. |

Окончание таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | List = [1,2]  N = 1  Slided = \_  Left = [1]  Right = [2] | 1 length( [2], 1)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Объединение списков Right и Left. |
| 16 | L = [1] | 1 append([2], [1], \_)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Вход в рекурсию. |
| 17 | X = \_  M = \_  L = [1]  N = \_ | 1 append([2], [1], \_)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | - |
| 18 | List = [1,2]  N = 1  Slided = [2,1]  Left = [1]  Right = [2] | 1 append([2], [1], \_)  2 shift\_right([1,2], 1, \_)  3 \_PROLOG\_Goal() | Объединение получено. |
| 19 | R = [2,1] | 1 \_PROLOG\_Goal() | Конец трассы. |

**ПРИЛОЖЕНИЕ Г**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ ДЛИНУ СПИСКА БЕЗ ОПРЕДЕЛЕННОГО ЭЛЕМЕНТА, GUI ПРИЛОЖЕНИЕ**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) My Company

Project: GUI\_LAB3

FileName: GUI\_COUNT.PRO

Purpose: No description

Written by: Visual Prolog

Comments:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

include "gui\_lab3.inc"

include "gui\_lab3.con"

include "hlptopic.con"

%BEGIN\_WIN Count

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Creation and event handling for window: Count

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

domains

intlist=char\*

num=char

length=integer

str=string

constants

%BEGIN Count, CreateParms, 14:27:21-23.10.2019, Code automatically updated!

win\_count\_WinType = w\_TopLevel

win\_count\_Flags = [wsf\_SizeBorder,wsf\_TitleBar,wsf\_Maximize,wsf\_Minimize,wsf\_Close,wsf\_ClipSiblings,wsf\_ClipChildren]

win\_count\_RCT = rct(100,80,486,261)

win\_count\_Menu = no\_menu

win\_count\_Title = "Count"

win\_count\_Help = idh\_contents

%END Count, CreateParms

predicates

len(intlist,num,length)

string\_namelist(str,intlist)

win\_count\_eh : EHANDLER

clauses

len([],T,0).

len([H|T],N,L) :- H <> N, len(T,N,L1), L = L1 + 1.

len([H|T],N,L) :- H = N, len(T,N,L).

string\_namelist(S,[H|T]):- frontchar(S,H,S1),!, string\_namelist(S1,T).

string\_namelist(\_,[]).

win\_count\_Create(\_Parent):-

win\_Create(win\_count\_WinType,win\_count\_RCT,win\_count\_Title,

win\_count\_Menu,\_Parent,win\_count\_Flags,win\_count\_eh,0).

%BEGIN Count, e\_Create

win\_count\_eh(\_Win,e\_Create(\_),0):-!,

%BEGIN Count, InitControls, 14:27:21-23.10.2019, Code automatically updated!

win\_CreateControl(wc\_PushButton,rct(274,140,370,164),"Ðåçóëüòàò",\_Win,[wsf\_Group,wsf\_TabStop],idc\_push\_button),

win\_CreateControl(wc\_Text,rct(15,33,118,57),"Ñïèñîê",\_Win,[wsf\_AlignLeft],idct\_ñïèñîê),

win\_CreateControl(wc\_Text,rct(15,64,118,88),"Ýëåìåíò",\_Win,[wsf\_AlignLeft],idct\_îòñóòñòâóþùèé\_ýëåìåíò),

win\_CreateControl(wc\_Text,rct(15,96,118,131),"Äëèíà ñïèñêà áåç ýëåìåíòà",\_Win,[wsf\_AlignLeft],idct\_äëèíà\_ñïèñêà\_áåç\_ýëåìåíòà),

win\_CreateControl(wc\_Edit,rct(140,33,370,57),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit1),

win\_CreateControl(wc\_Edit,rct(140,64,370,88),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit2),

win\_CreateControl(wc\_Edit,rct(140,100,370,124),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit3),

win\_CreateControl(wc\_Text,rct(15,140,217,160),"Ââîäèòü ñïèñîê áåç ïðîáåëîâ",\_Win,[wsf\_AlignLeft],idct\_ââîäèòü\_ñïèñîê\_áåç\_ïðîáåëîâ),

%END Count, InitControls

%BEGIN Count, ToolbarCreate, 14:27:21-23.10.2019, Code automatically updated!

%END Count, ToolbarCreate

!.

%END Count, e\_Create

%MARK Count, new events

%BEGIN Count, e\_Size

win\_count\_eh(\_Win,e\_Size(\_Width,\_Height),0):-!,

ifdef use\_tbar

toolbar\_Resize(\_Win),

enddef

!.

%END Count, e\_Size

%BEGIN Count, e\_Menu, Parent window

win\_count\_eh(Win,e\_Menu(ID,CAS),0):-!,

PARENT = win\_GetParent(Win),

win\_SendEvent(PARENT,e\_Menu(ID,CAS)),

!.

win\_count\_eh(\_Win,e\_Control(idc\_push\_button,\_CtrlType,\_CtrlWin,\_CtlInfo),0):-

!, Ctl1=win\_GetCtlHandle(\_Win,idc\_edit1),

String1 = win\_GetText(Ctl1),

string\_namelist(String1, X),

Ctl2=win\_GetCtlHandle(\_Win,idc\_edit2),

String2 = win\_GetText(Ctl2),

str\_char(String2,Y),

len(X, Y, Z),

str\_int(String,Z),

Ctl3=win\_GetCtlHandle(\_Win,idc\_edit3),

win\_SetText(Ctl3,String),

!.

%END Count, e\_Menu, Parent window

%END\_WIN Count

**ПРИЛОЖЕНИЕ Д**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ ДЛИНУ СПИСКА БЕЗ ОПРЕДЕЛЕННОГО ЭЛЕМЕНТА, КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ**

domains

integerlist = char\*

length = integer

number = char

predicates

len(integerlist,number,length).

clauses

len([],El,0).

len( [ Head | Tail ],El,N) :- Head <> El, len(Tail,El,N1), N = N1 + 1.

len( [ Head | Tail ],El,N) :- Head = El, len(Tail,El,N).

goal

len([1,2,1,4,5],1,L).

**ПРИЛОЖЕНИЕ Е**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ПРОГРАММЫ ПОИСКА ДЛИНЫ СПИСКА**

Трасса выполнения запроса поиска длины списка [1,2,3] без символа 1 представлена в табл. 2.

Таблица 2 − Трасса выполнения поиска длины

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Переменная | Стек вызовов | Пояснение работы |
| 1 | L = \_ | 1 \_PROLOG\_Goal() | Просмотр цели prolog-программы |
| 2 | El = | 1 len([,,], , \_)  2 \_PROLOG\_Goal() | Добавление цели в Call Stack. |
| 3 | Head = \_  Tail = \_  El = \_  N = \_  N1 = \_ | 1 len([,,], , \_)  2 \_PROLOG\_Goal() | Просмотр правила когда первый элемент списка не равен El |
| 4 | Head = \_  Tail = \_  El = \_  N = \_ | 1 len([,,], , \_)  2 \_PROLOG\_Goal() | Просмотр правила когда первый элемент списка равен El |
| 5 | L = 2 | 1 \_PROLOG\_Goal() | Длина списка без El равна 2 |

**ПРИЛОЖЕНИЕ Ж**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ РАЗНОСТЬ ДВУХ СПИСКОВ, GUI ПРИЛОЖЕНИЕ**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) My Company

Project: GUI\_LAB3

FileName: GUI\_SUB.PRO

Purpose: No description

Written by: Visual Prolog

Comments:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

include "gui\_lab3.inc"

include "gui\_lab3.con"

include "hlptopic.con"

%BEGIN\_WIN Sub

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Creation and event handling for window: Sub

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

domains

list = char\*

constants

%BEGIN Sub, CreateParms, 22:30:30-22.10.2019, Code automatically updated!

win\_sub\_WinType = w\_TopLevel

win\_sub\_Flags = [wsf\_SizeBorder,wsf\_TitleBar,wsf\_Maximize,wsf\_Minimize,wsf\_Close,wsf\_ClipSiblings,wsf\_ClipChildren]

win\_sub\_RCT = rct(100,80,497,270)

win\_sub\_Menu = no\_menu

win\_sub\_Title = "Sub"

win\_sub\_Help = idh\_contents

%END Sub, CreateParms

predicates

sub(list, list, list)

enter(char, list)

string\_namelist(string,list)

namelist\_string(list,string)

win\_sub\_eh : EHANDLER

clauses

enter(X, [X|L]).

enter(X, [Y|L]) :- enter(X,L).

sub([], L, []).

sub([X|M], L, N) :- enter(X,L), sub(M, L, N).

sub([X|M], L, [X|N]) :- not(enter(X,L)), sub(M, L, N).

string\_namelist(S,[H|T]):- frontchar(S,H,S1),!, string\_namelist(S1,T).

string\_namelist(\_,[]).

namelist\_string([], ""):-!.

namelist\_string([Head|ListTail], String):- namelist\_string(ListTail, StringTail), frontchar(String, Head, StringTail).

win\_sub\_Create(\_Parent):-

win\_Create(win\_sub\_WinType,win\_sub\_RCT,win\_sub\_Title,

win\_sub\_Menu,\_Parent,win\_sub\_Flags,win\_sub\_eh,0).

%BEGIN Sub, e\_Create

win\_sub\_eh(\_Win,e\_Create(\_),0):-!,

%BEGIN Sub, InitControls, 22:30:30-22.10.2019, Code automatically updated!

win\_CreateControl(wc\_PushButton,rct(269,153,365,177),"Ðåçóëüòàò",\_Win,[wsf\_Group,wsf\_TabStop],idc\_push\_button),

win\_CreateControl(wc\_Text,rct(16,99,112,135),"Ðàçíîñòü ñïèñêîâ X è Y",\_Win,[wsf\_AlignLeft],idct\_static\_text),

win\_CreateControl(wc\_Text,rct(16,64,112,84),"Ñïèñîê Y",\_Win,[wsf\_AlignLeft],idct\_static\_text1),

win\_CreateControl(wc\_Text,rct(16,28,112,48),"Ñïèñîê Õ",\_Win,[wsf\_AlignLeft],idct\_static\_text2),

win\_CreateControl(wc\_Edit,rct(150,28,364,52),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit1),

win\_CreateControl(wc\_Edit,rct(150,64,364,88),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit2),

win\_CreateControl(wc\_Edit,rct(150,102,364,126),"",\_Win,[wsf\_Group,wsf\_TabStop,wsf\_AutoHScroll,wsf\_AlignLeft],idc\_edit3),

%END Sub, InitControls

%BEGIN Sub, ToolbarCreate, 22:30:30-22.10.2019, Code automatically updated!

%END Sub, ToolbarCreate

!.

%END Sub, e\_Create

%MARK Sub, new events

%BEGIN Sub, e\_Size

win\_sub\_eh(\_Win,e\_Size(\_Width,\_Height),0):-!,

ifdef use\_tbar

toolbar\_Resize(\_Win),

enddef

!.

%END Sub, e\_Size

%BEGIN Sub, e\_Menu, Parent window

win\_sub\_eh(Win,e\_Menu(ID,CAS),0):-!,

PARENT = win\_GetParent(Win),

win\_SendEvent(PARENT,e\_Menu(ID,CAS)),

!.

win\_sub\_eh(\_Win,e\_Control(idc\_push\_button,\_CtrlType,\_CtrlWin,\_CtlInfo),0):-

!, Ctl1=win\_GetCtlHandle(\_Win,idc\_edit1),

String1 = win\_GetText(Ctl1),

string\_namelist(String1, X),

Ctl2=win\_GetCtlHandle(\_Win,idc\_edit2),

String2 = win\_GetText(Ctl2),

string\_namelist(String2, Y),

sub(X, Y, Z),

namelist\_string(Z,String),

Ctl3=win\_GetCtlHandle(\_Win,idc\_edit3),

win\_SetText(Ctl3,String),

!.

%END Sub, e\_Menu, Parent window

%END\_WIN Sub

**ПРИЛОЖЕНИЕ И**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ РАЗНОСТЬ ДВУХ СПИСКОВ, КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ**

domains

list = symbol\*

predicates

sub(list, list, list)

enter(symbol, list)

clauses

enter(X, [X|L]).

enter(X, [Y|L]) :- enter(X,L).

sub([], L, []).

sub([X|M], L, N) :- enter(X,L), sub(M, L, N).

sub([X|M], L, [X|N]) :- not(enter(X,L)), sub(M, L, N).

goal

sub([a, b, c], [a, e, f], N).

**ПРИЛОЖЕНИЕ К**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ПРОГРАММЫ РАЗНОСТИ ДВУХ СПИСКОВ**

Трасса выполнения запроса разности списка [a,b] и списка [a] представлена в табл. 3.

Таблица 3 − Трасса выполнения поиска разности двух списков

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Переменная | Стек вызовов | Пояснение работы |
| 1 | N = \_ | 1 \_PROLOG\_Goal() | Просмотр цели prolog-программы |
| 2 | L = ["a"] | 1 sub([a, b], [a], \_)  2 \_PROLOG\_Goal() | Добавление цели в Call Stack. Проход по правилам sub. |
| 3 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub([a, b], [a], \_)  2 \_PROLOG\_Goal() | Просмотр второго правила sub |
| 4 | X = "a"  L = \_ | 1 enter("a",["a"])  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Просмотр правила enter. Добавление правила в Call Stack.  "а" содержится в списке ["a"] |
| 5 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub(["a", "b"], ["a"],\_)  2 \_PROLOG\_Goal() | Удаление "а" из "b".  Удаление правила из Call Stack. |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | L=["a"] | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Поиск разности списков ["b"] и ["a"] |
| 7 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Просмотр второго правила sub |
| 8 | X = " b "  L = \_ | 1 enter("b",["a"])  2 sub(["b"], ["a"], \_)  3 sub(["a", "b"], ["a"],\_)  4 \_PROLOG\_Goal() | Просмотр правила enter. Добавление правила в Call Stack. |
| 9 | X = " b "  Y = \_  L = \_ | 1 enter("b",["a"])  2 sub(["b"], ["a"], \_)  3 sub(["a", "b"], ["a"],\_)  4 \_PROLOG\_Goal() | Вход в рекурсию enter. "b" не содержится в списке ["a"]. |
| 10 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Просмотр третьего правила sub |

Окончание таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | X = " b "  L = \_ | 1 enter("b",["a"])  2 sub(["b"], ["a"], \_)  3 sub(["a", "b"], ["a"],\_)  4 \_PROLOG\_Goal() | Просмотр правила enter. Добавление правила в Call Stack. |
| 12 | X = " b "  Y = \_  L = \_ | 1 enter("b",["a"])  2 sub(["b"], ["a"], \_)  3 sub(["a", "b"], ["a"],\_)  4 \_PROLOG\_Goal() | Вход в рекурсию enter. "b" не содержится в списке ["a"]. |
| 13 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Удаление правила из Call Stack. |
| 14 | L=["a"] | 1 sub([], ["a"], \_)  2 sub(["b"], ["a"], \_)  3 sub(["a", "b"], ["a"],\_)  4 \_PROLOG\_Goal() | Просмотр первого правила sub. |
| 15 | X = \_  M = \_  L = ["a"]  N = \_ | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Удаление правила из Call Stack. |
| 16 | X = \_  M = \_  L = ["a"]  N = ["b"] | 1 sub(["b"], ["a"], \_)  2 sub(["a", "b"], ["a"],\_)  3 \_PROLOG\_Goal() | Просмотр второго правила sub. |
| 17 | N = ["b"] | 1 \_PROLOG\_Goal() | Результат список ["b"] |