Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №19

По предмету: «Функциональное и логическое программирование»

Студент: Лаврова А. А.,
Группа: ИУ7-65Б
Преподаватель: Толпинская Н. Б.
Строганов Ю. В.

Москва, 2020 г

Задание:

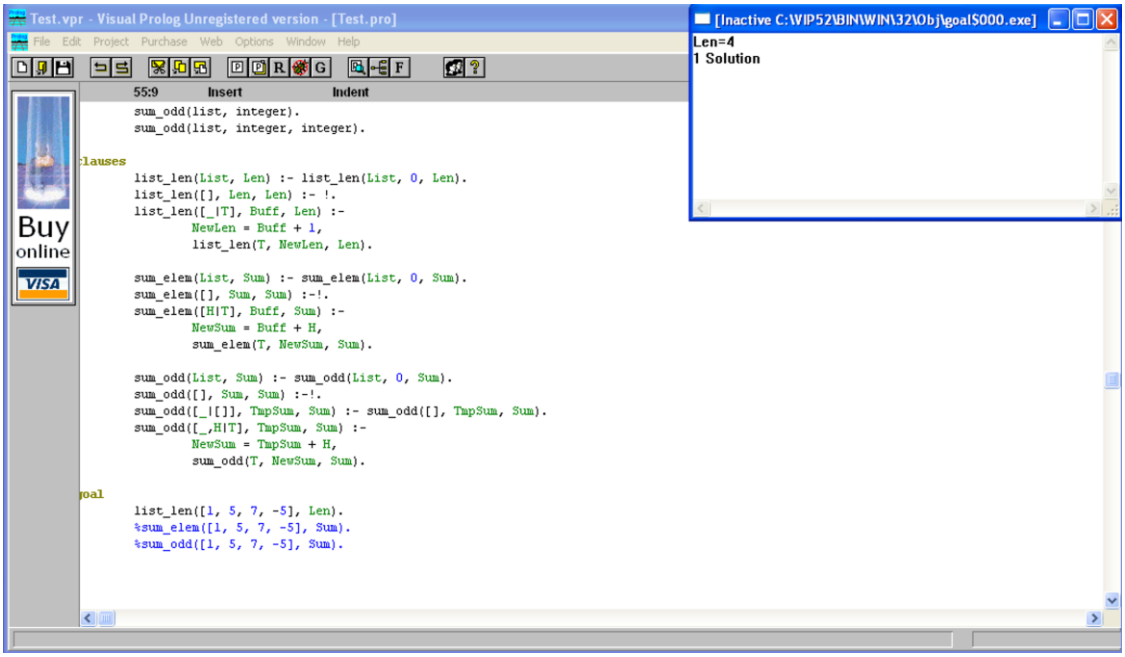
Используя хвостовую рекурсию, разработать эффективную программу, (комментируя назначение аргументов), позволяющую:

1. Найти длину списка (по верхнему уровню);
2. Найти сумму элементов числового списка
3. Найти сумму элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0)

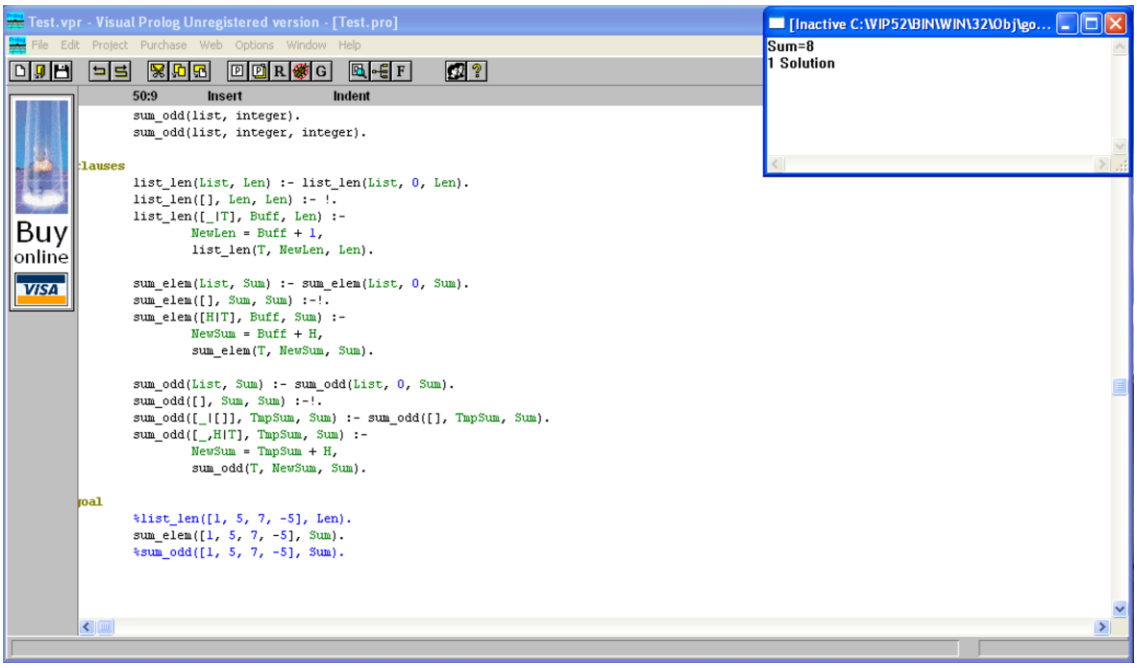
Листинг программы:

```
domains
        list = integer*.

predicates
        list_len(list, integer).
        list_len(list, integer, integer).

        sum_elem(list, integer).
        sum_elem(list, integer, integer).

        sum_odd(list, integer).
        sum_odd(list, integer, integer).

clauses
        /* задание 1 - вычисление длины списка */
        list_len(List, Len) :- list_len(List, 0, Len).
        list_len([], Len, Len) :- !.
        list_len([_|T], Buff, Len) :-
                NewLen = Buff + 1,
                list_len(T, NewLen, Len).

       /* задание 2 - вычисление суммы элементов списка */
        sum_elem(List, Sum) :- sum_elem(List, 0, Sum).
        sum_elem([], Sum, Sum) :-!.
        sum_elem([H|T], Buff, Sum) :-
                NewSum = Buff + H,
                sum_elem(T, NewSum, Sum).

       /* задание 3 - вычисление суммы элементов списка, стоящих на нечетных
местах */
        sum_odd(List, Sum) :- sum_odd(List, 0, Sum).
        sum_odd([], Sum, Sum) :-!.
        sum_odd([_|[]], TmpSum, Sum) :- sum_odd([], TmpSum, Sum).
        sum_odd([_,H|T], TmpSum, Sum) :-
                NewSum = TmpSum + H,
                sum_odd(T, NewSum, Sum).

goal
        %list_len([1, 5, 7, -5], Len).
        %sum_elem([1, 5, 7, -5], Sum).
        sum_odd([1, 5, 7, -5], Sum).
```

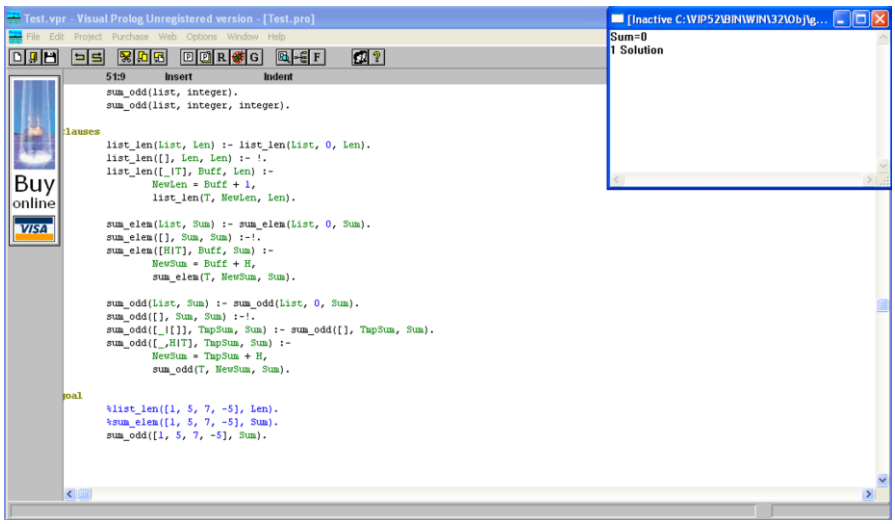
Результаты работы программы:



Вычисление длины списка



Вычисление суммы элементов списка



Вычисление суммы элементов списка, стоящих на нечётных местах

Описание аргументов:

1) list_len(list, integer).

   list – список для обработки; integer – результирующая длина списка

2) list_len(list, integer, integer).

   list – список для обработки; integer – текущая длина списка; integer – результирующая длина списка

3) sum_elem(list, integer).

   list – список для обработки; integer – результирующая сумма элементов списка

4) sum_elem(list, integer, integer).

   list – список для обработки; integer – текущая сумма; integer – результирующая сумма элементов списка

5) sum_odd(list, integer).

   list – список для обработки; integer – результирующая сумма элементов списка, стоящих на нечетных позициях

6) sum_odd(list, integer, integer).

7) list – список для обработки; integer – текущая сумма; integer – результирующая сумма элементов списка, стоящих на нечетных позициях

Работа с таблицей:

```
sum_elem(List, Sum) :- sum_elem(List, 0, Sum).
sum_elem([], Sum, Sum) :-!.
sum_elem([H|T], Buff, Sum) :-
                NewSum = Buff + H,
                sum_elem(T, NewSum, Sum).
```

…

```
sum_elem([10], Sum).
```

№ шага	Текущая резольвента – ТР	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
1	sum_elem([10], Sum).	ТЦ: sum_elem([10], Sum).	Поиск знания
2	sum_elem([10], Sum).	ТЦ: sum_elem([10], Sum). Сравниваемые термы: sum_elem([1, 5], Sum). sum_elem(List, Sum) Результат: успех Подстановка: List = [10], Sum=Sum	Проверка тела процедуры
3	sum_elem([10], 0, Sum).	ТЦ: sum_elem([10], 0, Sum). Сравниваемые термы: sum_elem([10], 0, Sum). sum_elem(List, Sum) Результат: неудача	Возврат к ТЦ, метка переносится ниже.
4	sum_elem([10], 0, Sum).	ТЦ: sum_elem([10], 0, Sum). Сравниваемые термы: sum_elem([10], 0, Sum). sum_elem([], Sum, Sum) Результат: унификация невозможна	Возврат к ТЦ, метка переносится ниже.
5	sum_elem([10], 0, Sum).	ТЦ: sum_elem([10], 0, Sum). Сравниваемые термы: sum_elem([10], 0, Sum). sum_elem([H|T], Buff, Sum Результат: успех Подстановка: H = 10, T = [],	Проверка тела процедуры

		Buff = 0, Sum=Sum	
6	NewSum = 0 + 1; sum_elem([], NewSum, Sum)	ТЦ: NewSum = 0 + 10 Результат: успех, конкретизация NewSum. Подстановка: NewSum=10	Переход к следующей цели.
7	sum_elem([], 10, Sum)	ТЦ: sum_elem([], 10, Sum)	Поиск знания
8	sum_elem([], 10, Sum)	ТЦ: sum_elem([], 10, Sum) Сравниваемые термы: sum_elem([], 10, Sum) Sum_elem(List, Sum) Результат: унификация невозможна	Возврат к ТЦ, метка переносится ниже.
9	sum_elem([], 10, Sum)	ТЦ: sum_elem([], 10, Sum) Сравниваемые термы: sum([], 10, Sum) Sum_elem([], Sum, Sum). Результат: успех Подстановка: 10 = Sum, Sum = 10	Проверка тела процедуры.
10	-	ТЦ: - Выполнение отсечения	Завершение работы программы Вывод «Sum = 10»

Вывод:

Эффективность работы программы достигнута за счет использования отсечения, которое ограничивает количество избыточных вычислений. Также использована хвостовая рекурсия, которая помогает оптимизировать использование памяти.

Теоретическая часть

1) *Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как можно организовать выход из рекурсии в Prolog?*
   Рекурсия – это ссылка при описании объекта на уже описанный объект. При описании знания – ссылка на то же знание.
   Для осуществления хвостовой рекурсии рекурсивный вызов определяемого предиката должен быть последней подцелью в теле рекурсивного правила и к моменту рекурсивного вызова не должно остаться точек возврата (непроверенных альтернатив). Параметры должны изменяться на каждом шаге так, чтобы в итоге либо сработал базис рекурсии, либо условие выхода из рекурсии, размещенное в самом правиле.

2) *Какое первое состояние резольвенты?*
   Заданный вопрос.

3) *В каких пределах программы переменные уникальны?*
   Переменные уникальны только в пределах предложения. Исключением является анонимные переменные.

4) *В какой момент, и каким способом системе удается получить доступ к голове списка?*
   Получить голову или хвост списка можно при унификации списка с [H|T], H – голова списка, T – хвост списка.

5) *Каково назначение использования алгоритма унификации?*
   Для поиска ответа на вопрос системе необходимо найти подходящее знание в БЗ, для поиска такого знания используется алгоритм унификации. Формально, он помогает системе понять, что заголовок подошел: алгоритм попарно пытается сопоставить

термы (текущую цель и термы из БЗ) и построить для них общий пример (для этого используется подстановка).

6) *Каков результат работы алгоритма унификации?*
   Алгоритм унификации может завершиться «успехом» и «неудачей». В случае успеха результирующая ячейка будет содержать подстановку(наиболее общий унификатор).

7) *Как формируется новое состояние резольвенты?*
   Сначала из стека выбирается первая подцель, а затем замена подцели на тело подходящего правила. Потом к полученной конъюнкции применяется подстановка, то есть наибольший общий унификатор.

8) *Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?*
   Применение подстановки {X1=T1, ..., Xn=Tn} заключается в замене каждого вхождения переменной Xi на соответствующий терм Ti. В результате применения подстановки некоторые переменные конкретизируются значениями, которые (значения) могут и будут далее использованы при доказательстве истинности тела выбранного правила.

9) *В каких случаях запускается механизм отката?*
   Откат происходит в случае тупиковой ситуации или в случае, если резольвента пуста.

10) *Когда останавливается работа системы? Как это определяется на формальном уровне?*
    Система завершает работу в случае, если найдены все возможные

ответы на вопрос. На формальном уровне – если в резольвенте находится исходный вопрос, для которого пройдена вся БЗ.