Лабораторная 1

Сравнение циклов и рекурсии

Цели

Оценить недостатки процедурного программирования
Научиться строить рекурсивные алгоритмы

Порядок выполнения

1. Написать программу по заданию с использованием цикла
2. Провести трассировку программы
3. Составить рекурсивную функцию для решения выданного задания
4. Реализовать составленную рекурсивную функцию на языке программирования
5. Написать отчет

Рекомендации по выполнению

Массивы фиксированной длины
Трассировка отключается макросом
Данные задаются внутри исходного кода

Состав отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Листинг программы с циклом
Результаты трассировки
Рекурсивная функция
Рекурсивная программа
Вводимые команды

Варианты заданий

1. Подсчитать сумму элементов массива
2. Подсчитать произведение элементов массива
3. Найти максимальное значение в массиве
4. Найти минимальное значение в массиве
5. Найти число нулевых элементов в массиве
6. Реализовать сложение через операцию увеличения на единицу
7. Реализовать умножение через сложение
8. Реализовать возведение в степень через умножение
9. Определить n-ное число Фибоначи
10. Определить факториал от числа n
11. Найти ближайшее снизу к n четное число
12. Найти ближайшее снизу к n нечетное число
13. Найти сумму чисел от 0 до n
14. Найти сумму четных чисел от 0 до n
15. Найти сумму нечетных чисел от 0 до n
16. Найти число четных чисел от 0 до n

Пример выполнения

Найти число нечетных чисел от 0 до n

```
main()
{
 int i, s=0, n=8;

 for(i=0; i<=n; i++ )
 {
  if( i%2 == 1 ) s++;
#if TRAS == 1
  printf("i=%d s=%d\n", i, s);
#endif
 }
 printf("s=%d\n", s);
}

cc -o loop loop.c -DTRAS=1
./loop
```

f(0)=0
f(x)=f(x-1), x-четно
f(x)=f(x-1)+1, x-нечетно

```
main()
{
 printf("s=%d\n", f(11) );
}
f(int i)
{
 if( i   == 0 ) return( 0 );
 if( i%2 == 1 ) return( f(i-1) + 1 );
 if( i%2 == 0 ) return( f(i-1) );
}
```

Лабораторная 2

S-выражения

Цели

Освоить S-выражения
Научиться основам работы в clisp
Познакомиться с функциями обработки списков

Порядок выполнения

1. Составить список1 по заданию в синтаксисе lisp
2. Написать функции для получения каждого из элементов списка1
3. Написать функцию для получения списка2
4. Написать функцию для получения списка3
5. Написать отчет

Рекомендации по выполнению

Сохраняйте удачные действия в отдельно текстовом файле по мере выполнения
Попробуйте перед выполнением основного задания поработать с простыми списками

Состав отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Полученные в ходе выполнения лабораторной работы функции
Результаты вызова функций

Варианты заданий

1. Список1={1,{2,3,4},5,{6,{7,8},9}} Список2={7,8,3} Список3={2,8,{3,4,1},6}
2. Список1={[1,2,{3,4,5,6},7},8,9} Список2={2,8,9} Список3={2,{8,3,4,1},6}
3. Список1={{{1,2,3,4},5,6,7,8},9} Список2={7,3,4} Список3={{7,8},{3,4},{1,6}}
4. Список1={1,2,{3,4,5},{{6,7,8,9}}} Список2={7,8,3} Список3={{2,8},6,7,8,9}
5. Список1={1,2,{3,4,5},{6,7,{8,9}}} Список2={7,8,9} Список3={2,8,{3,4,5},6}
6. Список1={1,{{2,3,4},5,6,7},8,9} Список2={7,8,3} Список3={2,{{8},3,4,1,6}}
7. Список1={1,{2,3,{4,5,6},7,8},9} Список2={7,8,3} Список3={2,{8,3,4,{1}},6}
8. Список1={{{1,2,3,4,}5,6,7,8},9} Список2={7,8,3} Список3={2,8,3,{4,1},6}
9. Список1={{{1,2,3,4,5,6,7,8},9}} Список2={7,8,3} Список3={2,8,3,4,{1,6}}
10. Список1={1,2,3,{4,{5,6,{7,8,9}}}} Список2={7,8,3} Список3={2,{8,3},4,1,6}
11. Список1={1,{2,3,{4,5,6,{7,8},9}}} Список2={7,8,3} Список3={2,8,3,4,{{1},6}}
12. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
13. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
14. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
15. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
16. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
17. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
18. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
19. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}
20. Список1={1,2,3,4,5,6,7,8,9} Список2={7,8,3} Список3={2,8,3,4,1,6}

Пример выполнения

Список1={{1,{2,3,4,5,6,7,8}},9} Список2={3,7,8} Список3={2,8,3,{4,1},6}

1. Составить список1 по заданию в синтаксисе lisp
[13]> '((1 (2 3 4 5 6 7 8)) 9)
((1 (2 3 4 5 6 7 8)) 9)
2. Написать функции для получения каждого из элементов списка1
[14]> (car '((1 (2 3 4 5 6 7 8)) 9))
(1 (2 3 4 5 6 7 8))
[15]> (car (car '((1 (2 3 4 5 6 7 8)) 9)))
1
(car (cdr (car '((1 (2 3 4 5 6 7 8)) 9))))
(2 3 4 5 6 7 8)
[17]> (car (car (cdr (car '((1 (2 3 4 5 6 7 8)) 9)))))
2
[18]> (setq a '((1 (2 3 4 5 6 7 8)) 9))
((1 (2 3 4 5 6 7 8)) 9)
[19]> (car (car (cdr (car a))))
2
[20]> (car (cdr (car (cdr (car a)))))
3
[21]> (car (cdr (cdr (car (cdr (car a))))))
4
.....
3. Написать функцию для получения списка2
[22]> (cons (car (cdr (car (cdr (car a))))) (cdr (cdr (cdr (cdr (cdr (car (cdr
(car a)))))))))
(3 7 8)
.....
4. Написать функцию для получения списка3
.....

Лабораторная 3

λ-выражения и β-редукция в λ-исчислении в языке lisp

Задания

Выполнить β-редукции несколькими способами
Составить программу на lisp для вычисления функции

Содержание отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Заданный пример
Последовательное получение бетта-редикса всеми возможными спсобами
Текст lisp-программы
Результаты вызова программы

Варианты заданий

1. ((((λxyz.x(yz))(λx.x+x))(λx.x*x))3)
2. ((((λxyz.x(yz))(λx.x*x))(λx.x+x))4)
3. ((((λxyz.x(yz))(λx.x*x))(λx.x*x))5)
4. ((((λxyz.x(yz))(λx.x+8))(λx.9*x))6)
5. ((((λxyz.x(yz))(λx.x))(λx.x*x))7)
6. ((((λxyz.x(yz))(λx.x+x))(λx.x+x))8)
7. ((((λxyz.xzy))(λxy.x/y))3)9)
8. ((((λxyz.xzy))(λxy.x/y))((λx.x)3))9)
9. ((((λxyz.xzy))(λxy.x/y))3)((λx.x)9))
10. ((((λxyz.xz(yz)))(λxy.x+y))(λx.x+2)3))
11. ((((λxyz.xz(yz)))(λxy.x))(λx.x)4))
12. ((((λxyz.xz(yz)))(λxy.y))(λx.x+2)5))
13. ((((λxyz.xz(yz)))(λxy.x*y))(λx.x+2)6))
14. ((((λxyz.xz(yz)))(λxy.x+y))(λx.x*2)7))
15. ((((λxyz.xz(yz)))(λxy.x-y))(λx.x-2)8))

Пример выполнения

(((λxyz.x(yz))(λx.x))(λx.x+x))2=((λyz.(λx.x)(yz))(λx.x+x))2=
(λz.(λx.x)((λx.x+x)z))2=

1) (λx.x)((λx.x+x)2)=(λx.x+x)2=2+2
2) (λz.((λx.x+x)z))2=(λz.z+z)2=2+2

```
> (
> (lambda (x y z) (x (y z)))
> (lambda (x) (* x 1))
```

```
> (lambda (x) (+ x x))
> 2)
4
```

Лабораторная 4

Рекурсивные функции в Lisp

Цели

Научиться составлять рекурсивные функции на языке Lisp

Порядок выполнения

1. Составить рекурсивную функцию для решения выданного задания
2. Реализовать составленную рекурсивную функцию на языке Lisp
3. Написать отчет

Рекомендации по выполнению

Изначально, рекурсивную функцию составить в нотации Хендерсона.
Для трансляции программы использовать clisp или scm. При использовании clisp не забывать использовать «funcall» при вызове подпрограмм.

Состав отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Рекурсивная функция
Название способа организации рекурсии
Программа на явыке Lisp
Результат вызова функции с двумя разными значениями аргументов

Варианты заданий

1. Подсчитать сумму элементов списка
2. Подсчитать произведение элементов списка
3. Найти максимальное значение в списке
4. Найти минимальное значение в списке
5. Найти число нулевых элементов в списке
6. Реализовать сложение через операцию увеличения на единицу
7. Реализовать умножение через сложение
8. Реализовать возведение в степень через умножение
9. Определить n-ное число Фибоначи
10. Определить факториал от числа n
11. Построить список, элементами которого являются суммы соответствующих элементов двух заданных списков
12. Построить список, элементами которого являются разности соответствующих элементов двух заданных списков
13. Найти сумму чисел от 0 до n
14. Найти сумму четных чисел от 0 до n
15. Найти сумму нечетных чисел от 0 до n
16. Найти число четных чисел от 0 до n

Примеры выполнения

Найти число нечетных чисел от 0 до n

Функция
summ2(x) = if x=0 then 0 else if odd(x) summ2(x-1)+1 else summ2(x-1)
Программа

```
(defun ssum2 (x)
 (if (= x 0) 0
             (if (oddp x) (+ (ssum2 (- x 1)) 1)
                          (ssum2 (- x 1))
             )
 )
)
```

Вызов 1
(ssum2 8)
4
Вызов 2
(ssum2 1)
1

Увеличить все элементы списка на 1

Функция
plus1(x) = if x = nil then nil
else cons(car(x)+1, plus1(cdr(x)))
Программа

```
(defun plus1 (x)
 (if x (cons (+ (car x) 1)
             (plus1 (cdr x))
       )
       nil
 )
)
```

Вызов 1
(plus1 '(1 9 6))
(2 10 7)
Вызов 2
(plus1 '(1 4))
(2 5)

Контрольные вопросы

Какие бывают способы организации рекурсии
Каким способом организуется рекурсия в задании N n
Что такое рекурсия
Почему в Lisp невозможен цикл
Что такое функция

Почему в Lisp пишется (car x), а не car(x)

Лабораторная 5

Функции высших порядков

Цели

Научиться составлять функции высших порядков на языке Lisp

Порядок выполнения

1. Определить состав задания по номеру варианта
2. Составить и реализовать на Lisp функции соответствующие заданию
3. Составить и реализовать на Lisp функцию высшего порядка соответствующую заданию
4. Написать отчет

Рекомендации по выполнению

Изначально, рекурсивную функцию составить в нотации Хендерсона.
При использовании clisp не забывать использовать «funcall» при вызове подпрограмм.
Попробуйте использовать другие функции в качестве аргумента.

Состав отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Функция высшего порядка
Программа на языке Lisp
Результат вызова функции с двумя разными значениями аргументов

Варианты заданий

Вариант	Тип списка	Отображение	Редукция
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	1
5	5	1	2
6	6	1	3
7	7	1	1
8	1	2	2
9	2	2	3
10	3	2	1
11	4	2	2
12	5	2	3
13	6	2	1
14	7	2	2

15	1	3	3
16	2	3	1
17	3	3	2
18	4	3	3
19	5	3	1
20	6	3	2
21	7	3	3

типы списка

1. ((x x) (x x) (x x) (x x) ...)
2. ((x x x) (x x x) (x x x) (x x x) ...)
3. ((x) (x) (x) (x) ...)
4. ((x x ...) (x x x ...))
5. ((x (x)) (x (x)) (x (x)) (x (x)) ...)
6. (x (x (x (x ...(x)))))
7. (((((x)... x) x) x) x)

отображение

1. Умножить все элементы списка на заданное число
2. Возвести в заданную степень все элементы списка
3. Определить, является ли заданная списком последовательность последовательностью Фибоначи

редукция

1. Найти максимальное значение в списке
2. Подсчитать произведение элементов списка
3. Найти число нулевых элементов в списке

Примеры выполнения

Увеличить все элементы списка на 1

Программа

```
(defun otobr(x f)
(if x (cons (funcall f (car x)) (otobr (cdr x) f))
nil)

(otobr '(7 3) (lambda (x) (+ x 1)))

(setq ft (lambda (x) (+  x 1)))

(otobr '(4 8) ft)
```

Контрольные вопрсы

Что такое функции высших порядков?

Какие знаете виды функций высших порядков?

Лабораторная 6

Основные возможности Haskell

Цели

Приобрести навыки работы с интерпретатором языка Haskell. Получить представление об основных типах языка Haskell. Научиться определять простейшие функции.

Порядок выполнения

1. Изучить теоретические сведения
2. Выполнить задания в соответствии с вариантом
3. Написать отчет

Рекомендации по выполнению

Выполните приведеные в теоретических сведениях примеры.

Состав отчета

Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
Программа на явыке Haskell
Результат вызова функции с двумя разными значениями аргументов

Теоретические сведения

Типы

• Типы Integer и Int используется для представления целых чисел, причем значения типа Integer не ограничены по длине.
• Типы Float и Double используется для представления вещественных чисел.
• Тип Bool содержит два значения: True и False, и предназначен для представления результата логических выражений.
• Тип Char используется для представления символов.

Списки

Чтобы задать список в Haskell, необходимо в квадратных скобках перечислить его элементы через запятую. Все эти элементы должны принадлежать одному и тому же типу. Тип списка с элементами, принадлежащими типу a, обозначается как [a].

```
>[1,2]
[1,2] :: [Integer]
>['1','2','3']
['1','2','3'] :: [Char]
```

Операции со списками

Оператор : (двоеточие) используется для добавления элемента в начало списка. Его левым аргументом должен быть элемент, а правым — список:

```
>1:[2,3]
[1,2,3] :: [Integer]
>'5':['1','2','3','4','5']
['5','1','2','3','4','5'] :: [Char]
>False:[]
[False] :: [Bool]
```

С помощью оператора (:) и пустого списка можно построить любой список:

```
>1:(2:(3:[]))
[1,2,3] :: Integer
```

Оператор (:) ассоциативен вправо, поэтому в приведенном выше выражении можно опустить скобки:

```
>1:2:3:[]
[1,2,3] :: Integer
```

Элементами списка могут быть любые значения — числа, символы, кортежи, другие списки и т.д.

```
>[(1,'a'),(2,'b')]
[(1,'a'),(2,'b')] :: [(Integer,Char)]
>[[1,2],[3,4,5]]
[[1,2],[3,4,5]] :: [[Integer]]
```

Для работы со списками в языке Haskell существует большое количество функций. В данной лабораторной работе рассмотрим только некоторые из них.

- Функция head возвращает первый элемент списка.
- Функция tail возвращает список без первого элемента.
- Функция length возвращает длину списка.

Функции head и tail определены для непустых списков. При попытке применить их к пустому списку интерпретатор сообщает об ошибке. Примеры работы с указанными функциями:

```
>head [1,2,3]
1 :: Integer
>tail [1,2,3]
[2,3] :: [Integer]
>tail [1]
[] :: Integer
>length [1,2,3]
3 :: Int
```

Для соединения (конкатенации) списков в Haskell определен оператор ++.

```
>[1,2]++[3,4]
[1,2,3,4] :: Integer
```

Функции

```
square :: Integer -> Integer
square x = x * x
```

Первая строка (square :: Integer -> Integer) объявляет, что

мы определяем функцию square, принимающую параметр типа Integer и возвращающую результат типа Integer. Вторая строка (square x = x * x) является непосредственно определением функции. Функция square принимает один аргумент и возвращает его квадрат.

тип функции, принимающей n аргументов, принадлежащих типам t1, t2, . . . , tn, и возвращающей результат типа a, записывается в виде t1->t2->...->tn->a.

```
add :: Integer -> Integer -> Integer
add x y = x + y
```

Условное выражение

if условие then выражение else выражение.

Запишем функцию signum, вычисляющую знак переданного ей аргумента:

```
signum :: Integer -> Integer
signum x = if x > 0 then 1
           else if x < 0 then -1
                else 0
```

Условие в определении условного оператора представляет собой любое выражение типа Bool. Примером таких выражений могут служить сравнения. При сравнении можно использовать следующие операторы:

- <, >, <=, >= — эти операторы имеют такой же смысл, как и в языке Си (меньше, больше, меньше или равно, больше или равно).
- == — оператор проверки на равенство.
- /= — оператор проверки на неравенство.

Варианты заданий

1) Функция max3, по трем целым возвращающая наибольшее из них.
2) Функция min3, по трем целым возвращающая наименьшее из них.
3) Функция sort2, по двум целым возвращающая пару, в которой наименьшее из них стоит на первом месте, а наибольшее — на втором.
4) Функция bothTrue :: Bool -> Bool -> Bool, которая возвращает True тогда и только тогда, когда оба ее аргумента будут равны True. Не используйте при определении функции стандартные логический операции (&&, || и т.п.).
5) Функция solve2::Double->Double->(Bool,Double), которая по двум числам, представляющим собой коэффициенты линейного уравнения ax + b = 0, возвращает пару, первый элемент которой равен True, если решение существует и False в противном случае; при этом второй элемент равен либо значению корня, либо 0.0.
6) Функция isParallel, возвращающая True, если два отрезка, концы которых задаются в аргументах функции, параллельны (или лежат на одной прямой). Например, значение выраже-

ния isParallel (1,1) (2,2) (2,0) (4,2) должно быть равно True, поскольку отрезки (1, 1) − (2, 2) и (2, 0) − (4, 2) параллельны.

7) Функция isIncluded, аргументами которой служат параметры двух окружностей на плоскости (координаты центров и радиусы); функция возвращает True, если вторая окружность целиком содержится внутри первой.
8) Функция isRectangular, принимающая в качестве параметров координаты трех точек на плоскости, и возвращающая True, если образуемый ими треугольник — прямоугольный.
9) Функция isTriangle, определяющая, можно ли их отрезков с заданными длинами x, y и z построить треугольник.
10) Функция isSorted, принимающая на вход три числа и возвращающая True, если они упорядочены по возрастанию или по убыванию.

Контрольные вопросы

1. В чем отличие команд интерпретатора от выражений языка Haskell?
2. Основные типы языка Haskell.
3. Функции для работы с кортежами.
4. Функции для работы со списками.
5. Допустимые имена переменных и функций.
6. Команды интерпретатора для работы с файлами программ.
7. Условные выражения в языке Haskell.
8. Определение функций в языке Haskell.

Лабораторная 7

Использование комбинаторов в языке Haskell

Задание

```
Построить из комбинаторов i, b, k, c, w, s
и функций p x = x + 1, u x y = x + y
выражение v, удовлетворяющее условиям варианта.
Функции p и u могут быть использованы не более одного раза
```

Варианты

1. v 6 7 = 7
2. v 6 = 8
3. v 5 = 10
4. v 4 = 9
5. v 3 5 = 9
6. v 4 7 = 5
7. v 3 4 5 = 9
8. v 7 2 6 = 2
9. v 8 2 = 3
10. v 8 7 = 14
11. v 7 9 = 14
12. v 2 = 4
13. v 9 = 18
14. v 5 = 11
15. v 2 6 = 9
16. v 3 5 = 4
17. v 8 2 5 = 7
18. v 7 3 6 = 3
19. v 8 1 = 2
20. v 8 6 = 12
21. v 4 9 = 8

Состав работы

1. Построить комбинаторы i, b, k, c, w, s
2. функции p, u
3. проверить их работоспособность
4. построить выражение в соответствии с заданием
5. доказать правильность решения

Пример выполнения

```
komb.hs
-------------
i x = x
b ...
k ...
...
-------------
ghci komb.hs
```

```
-------------
> i 5
5
> k i 6 7
7
-------------
K I 6 7 = I 7 = 7
```

Лабораторная 8

Формализация предметной области для языка Пролог

Задание

Для соответствующей варианту предметной области составить 3-5 аксиом и вопрос к ним. Записать на языке пролог.

Варианты

1. книжный шкаф
2. зоопарк
3. театр
4. цирк
5. телевизор
6. баня
7. таблица умножения
8. сдача экзамена
9. ноутбук
10. файловая система
11. правительства
12. система прав пользователей
13. продуктовый магазин
14. учебная аудитория
15. родственники
16. структура каталогов
17. арифметические действия
18. текст
19. гипертекст
20. холодильник
21. лес
22. огород

Состав работы

1. Составить предикаты
2. Записать аксиомы
3. Сформулировать вопрос
4. Записать на языке Пролог
5. Написать отчет

Пример выполнения

Предметная область: раковина
Описание предметной области:
Раковина засоряется тем, что пропустит фильтр.
Фильтр пропускает очистки морковки при использовании любых инструментов для чистки и очистки любого овощя очищенного картофелечисткой. Морковь почистили ножом, картофель - картофелечисткой.
Чем засорится раковина?
Засорится ли раковина?
Предикаты:
C(x,y) – чистить икс игреком
F(x) – фильтр пропустит x

R(x) — раковина засорилась x

A1: F(x) -> R(x)
A2: C(x, kartochistka) -> F(x)
A3: C(morkva, y) -> F(morkva)
A4: C(morkva, nozh)
A5: C(kartofan, kartochistka)
B1: R(x)
B2: R(kartofan)

rack.pl

```
r(X) :- f(X).
f(X) :- c(X, 'kartochistka').
f('morkva') :- c('morkva', _).
c('morkva', 'nozh').
c('kartofan', 'kartochistka').
```

```
$ gprolog
GNU Prolog 1.2.18
By Daniel Diaz
Copyright (C) 1999-2004 Daniel Diaz
| ?- consult('rack.pl').
compiling /home/kdb/prolog/rack.pl for byte code...
/home/kdb/prolog/rack.pl compiled, 5 lines read - 739 bytes written, 25 ms

yes
| ?- r('morkva').

yes
| ?- r(X).

X = kartofan ? a

X = morkva

yes
| ?- trace.
The debugger will first creep -- showing everything (trace)

yes

{trace}
| ?- r('kartofan').
      1    1  Call: r(kartofan) ?
      2    2  Call: f(kartofan) ?
      3    3  Call: c(kartofan,kartochistka) ?
      3    3  Exit: c(kartofan,kartochistka) ?
      2    2  Exit: f(kartofan) ?
      1    1  Exit: r(kartofan) ?

yes
{trace}
| ?- r(X).
      1    1  Call: r(_16) ?
      2    2  Call: f(_16) ?
      3    3  Call: c(_16,kartochistka) ?
      3    3  Exit: c(kartofan,kartochistka) ?
      2    2  Exit: f(kartofan) ?
      1    1  Exit: r(kartofan) ?

X = kartofan ?  a
```

```
      1    1  Redo: r(kartofan) ?
      2    2  Redo: f(kartofan) ?
      3    3  Call: c(morkva,_69) ?
      3    3  Exit: c(morkva,nozh) ?
      2    2  Exit: f(morkva) ?
      1    1  Exit: r(morkva) ?

X = morkva

yes
{trace}
| ?-
```

Лабораторная 9

Обработка списков на языке Пролог

Задание

Составить программу для обработки списка согласно варианту.

Варианты

1. Проверка наличия элемента в списке
2. Определение длины списка
3. нахождение наибольшего элемента в списке
4. Определение сортированности списка
5. Нахождение суммы элементов списка
6. Создание списка из чисел, находящихся в диапазоне между двумя заданными
7. Удаление элемента из списка

Состав работы

1. Составить предикаты
2. Записать аксиомы
3. Сформулировать вопрос
4. Написать отчет

Пример выполнения

sp.pl

```
app([],L2,L2).
app( [X | L1], L2, [X | L3]):-app( L1, L2, L3).
```

```
$ gprolog
GNU Prolog 1.2.18
By Daniel Diaz
Copyright (C) 1999-2004 Daniel Diaz
| ?- consult('sp.pl').
compiling /home/kdb/prolog/rack.pl for byte code...
/home/kdb/prolog/sp.pl compiled, 5 lines read - 739 bytes written, 25 ms

yes
| ?- app( [a, b, c], [1, 2, 3], L ).
L = [a, b, c, 1, 2, 3]

yes
| ?- app( L, [1, 2, 3], [a, b, c, 1, 2, 3] ).
L = [a, b, c]
```