**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования   
«Тульский государственный университет»**

**Интернет-институт**

ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ

по дисциплине

«Интеллектуальные системы в промышленности»

Семестр 6

Вариант 3

Выполнил: студент гр. ИБ262521-ф

Артемов Александр Евгеньевич

Проверил: канд. техн. наук, доц.

Французова Юлия Вячеславовна

Тула 2024

**Практическая работа № 1.**

**Название работы:** Отладка рекурсивных программ на языке Пролог.

**Цели работы:** Изучить и освоить на примерах использование механизма рекурсии в Прологе.

**Задание:**

1. Создайте предикат, вычисляющий неотрицательную степень целого числа.
2. Создать предикат, вычисляющий по натуральному числу N сумму чисел от 1 до N.
3. Создайте предикат, вычисляющий по натуральному числу N сумму нечетных чисел, не превосходящих N.
4. Создайте предикат, вычисляющий наибольший общий делитель двух натуральных чисел.
5. Создайте предикат, вычисляющий наименьшее общее кратное двух натуральных чисел.
6. Создайте предикат, вычисляющий отрицательную целую степень действительного числа.
7. Создайте предикат, вычисляющий функцию sin(x) с заданной точностью.
8. Создайте предикат, вычисляющий функцию sin(x), используя разложение в ряд по заданному количеству членов ряда.
9. Создайте предикат, вычисляющий по натуральному числу N сумму четных чисел, не превосходящих N.
10. Создайте предикат, вычисляющий функцию exp(x), используя разложение в ряд по заданному количеству членов ряда.
11. Создайте предикат, вычисляющий функцию exp (x), используя разложение в ряд по заданному количеству членов ряда.
12. Создайте предикат, вычисляющий функцию cos(x) с заданной точностью.
13. Создайте предикат, вычисляющий функцию cos(x), используя разложение в ряд по заданному количеству членов ряда.
14. Создайте предикат, вычисляющий рекурсивно значение , используя функцию квадратного корня.
15. Создать предикат, вычисляющий по натуральному числу N произведение чисел от (N div 2) до N.
16. Создайте предикат, вычисляющий по натуральному числу N среднеарифметическое натуральных четных чисел, не превосходящих N.
17. Создайте предикат, вычисляющий среднегеометрическое натуральных чисел, кратных 3, не превосходящих N.
18. Создайте предикат, выводящий простые числа на интервале (0, N). Использовать предикат, вычисляющий НОД.
19. Создайте предикат, выводящий все делители числа N.
20. Создайте предикат, вычисляющий функцию tg(x) с заданной точностью.
21. Создайте предикат, вычисляющий функцию tg(x), используя разложение в ряд по заданному количеству членов ряда.
22. Создайте предикат, вычисляющий по натуральному числу N произведение чисел, кратных 3 не превосходящих N.
23. Создайте предикат, вычисляющий функцию ln(x), используя разложение в ряд по заданному количеству членов ряда.
24. Создайте предикат, вычисляющий функцию ln(x), используя разложение в ряд по заданному количеству членов ряда.
25. Создайте предикат, вычисляющий функцию ctg(x) с заданной точностью.
26. Создайте предикат, вычисляющий функцию ctg(x), используя разложение в ряд по заданному количеству членов ряда.

**Выполнение практической работы.**

Изучены теоретические сведения практической работы о создании рекурсивных программ на языке Пролог.

1. Предикат, вычисляющий неотрицательную степень целого числа.

В качестве условия остановки рекурсии примем факт, что любое целое число в нулевой степени равно 1, т. е. *P0 = 1*. Таким образом, рекурсивно получаем, что *PN = PN-1× P*. Для вычисления неотрицательной степени целого числа определим предикат stepen(integer, integer, integer [out]), принимающий 1-ый и 2-ой аргументы как целое число и степень, в которую число необходимо возвести, соответственно, а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

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

stepen(P, N, R) :- N1 = N - 1, stepen(P, N1, R1), R = R1 \* P.

run() :-

stdio::write("Введите число: "), P = stdio::read(),

hasDomain(integer, P),

stdio::write("Введите степень: "), N = stdio::read(),

hasDomain(integer, N), stdio::write("Результат: "),

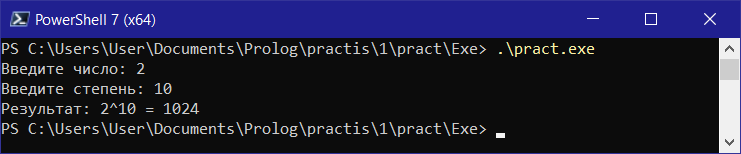
stepen(P, N, Res),

stdio::write(P), stdio::write("^"), stdio::write(N),

stdio::write(" = "), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката вводим число 2, степень 10 и получаем результат 210 = 1024.



1. Предикат, вычисляющий по натуральному числу N сумму чисел от 1 до N.

В качестве условия остановки рекурсии примем факт, что сумма чисел при *N = 1* равна 1. Таким образом, рекурсивно получаем, что сумма чисел от 1 до N равна *SN = SN-1 + N*. Для вычисления суммы чисел от 1 до N определим предикат mySum(integer, integer [out]), принимающий 1-ый аргумент как натуральное число, а 2-ой является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

mySum(1, 1) :- !.

mySum(N, R) :- N1 = N - 1, mySum(N1, R1), R = R1 + N.

run() :-

stdio::write("Введите натуральное число: "),

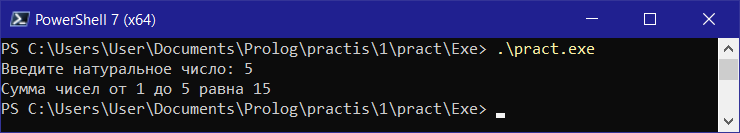
N = stdio::read(), hasDomain(integer, N),

stdio::write("Сумма чисел от 1 до "), stdio::write(N),

stdio::write(" равна "), mySum(N, Res), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката вводим число 5 и получаем сумму чисел от 1 до 5 *S5 = 1 + 2 + 3 + 4 + 5 = 15*.

1. Предикат, вычисляющий по натуральному числу N сумму нечетных чисел, не превосходящих N.

В качестве условия остановки рекурсии примем факт, что сумма чисел при *N = 1* равна 1. Таким образом, рекурсивно получаем, что сумма нечетных чисел, не превосходящих *N*, вычисляется как *SN = SN-1 + N* при нечетном N и как *SN = SN-1* при четном. Для вычисления суммы нечетных чисел, не превосходящих *N*, определим предикат myOddSum(integer, integer [out]), принимающий 1-ый аргумент как натуральное число *N*, а 2-ой является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

myOddSum(1, 1) :- !.

myOddSum(N, R) :-

myOddSum(N - 1, R1),

if N mod 2 = 0 then R = R1

else R = R1 + N

end if.

run() :-

stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N),

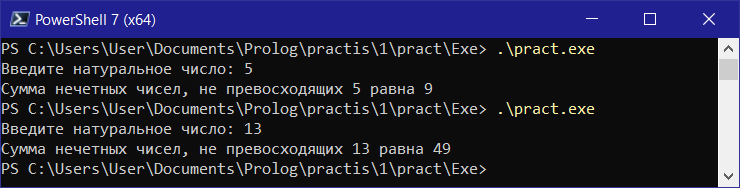
stdio::write("Сумма нечетных чисел, не превосходящих "),

stdio::write(N), stdio::write(" равна "),

myOddSum(N, Res), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката проверяем числа 5 и 13. Соответственно, при *N = 5 S5 = 1 + 3 + 5 = 9*, а при *N = 13 S13 = 1 + 3 + 5 + 7 + 9 + 11 + 13 = 49*.

1. Предикат, вычисляющий наибольший общий делитель двух натуральных чисел.

В качестве условия остановки рекурсии примем факт, что НОД натурального числа и 0 равен этому числу, т. е. *НОД(А, 0) = А*. Таким образом, рекурсивно получаем, что *НОД(А, В) = НОД(В, A % B)*. Для вычисления НОД определим предикат gcd(integer, integer, integer [out]), принимающий 2 аргумента как натуральные числа, а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

gcd(A, 0, A) :- !.

gcd(A, B, R) :- gcd(B, A mod B, R1), R = R1.

run() :-

stdio::write("Введите 1-ое число: "),

A = stdio::read(), hasDomain(integer, A),

stdio::write("Введите 2-ое число: "),

B = stdio::read(), hasDomain(integer, B),

stdio::write("НОД "), stdio::write(A),

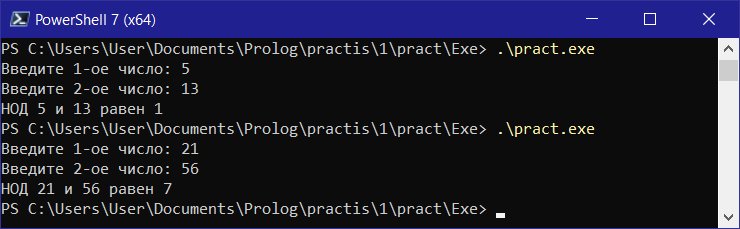
stdio::write(" и "), stdio::write(B),

stdio::write(" равен "),

gcd(A, B, Res), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката проверяем две пары чисел: простые 5 и 13, где НОД равен 1, а так же 21 и 56, где НОД равен 7.

1. Предикат, вычисляющий наименьшее общее кратное двух натуральных чисел.

Для вычисления НОК двух натуральных чисел воспользуемся таким фактом, что НОК равен произведению этих чисел поделенных нацело на их НОД, т. е. *НОК(А, В) = (А × В) / НОД(А, В)*. Для вычисления НОК используем предикат gcd/3 из 4-го задания для нахождения НОД, а так же определим предикат lcm(integer, integer, integer [out]), принимающий 2 аргумента как натуральные числа, а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

lcm(A, B, R) :- gcd(A, B, R1), R = A \* B div R1.

run() :-

stdio::write("Введите 1-ое число: "),

A = stdio::read(), hasDomain(integer, A),

stdio::write("Введите 2-ое число: "),

B = stdio::read(), hasDomain(integer, B),

stdio::write("НОД "), stdio::write(A),

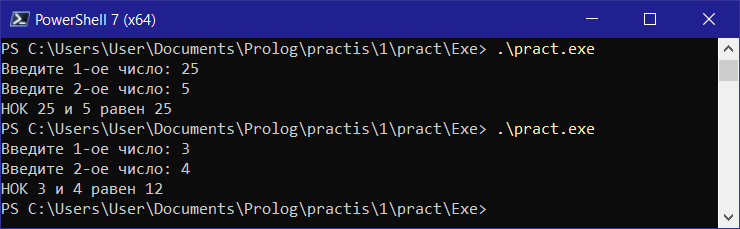
stdio::write(" и "), stdio::write(B),

stdio::write(" равен "),

lcm(A, B, Res), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката проверяем две пары чисел: 25 и 5, где НОК равен 25, а так же 3 и 4, где НОК равен 12.

1. Предикат, вычисляющий отрицательную целую степень действительного числа.

В качестве условия остановки рекурсии примем факт, что любое действительного число *R* в -1-ой степени равно 1/R, т. е. *R-1 = 1/R*. Таким образом, рекурсивно получаем, что . Для вычисления отрицательной степени целого числа определим предикат subZeroPow(real, integer, real [out]), принимающий 1-ый и 2-ой аргументы как действительное число и отрицательную степень, в которую необходимо возвести число, соответственно, а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

subZeroPow(A, -1, 1 / A) :- !.

subZeroPow(A, N, R) :- N1 = N + 1, subZeroPow(A, N1, R1), R = R1 \* (1 / A).

run() :-

stdio::write("Введите действительное число: "),

A = stdio::read(), hasDomain(real, A),

stdio::write("Введите отрицательную степень: "),

N = stdio::read(), hasDomain(integer, N),

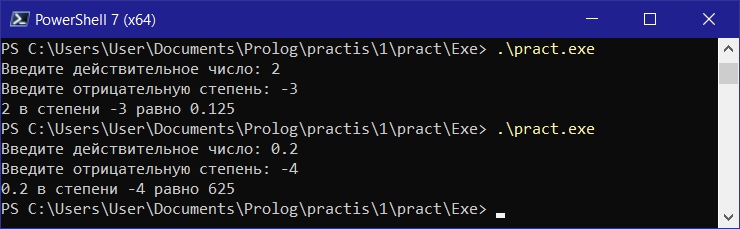
stdio::write(A), stdio::write(" в степени "),

stdio::write(N), stdio::write(" равно "),

subZeroPow(A, N, Res), stdio::write(Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем и .

1. Предикат, вычисляющий функцию sin(x) с заданной точностью.

Для вычисления функции sin(x) с заданной точностью используем формулу синуса тройного угла , откуда . В качестве условия остановки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,001. Для вычисления функции sin(x) с заданной точностью определим предикат sinus(real, real,   
real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить синус, 2-ой — заданная точность, а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

sinus(X, E, X) :- X < E, !.

sinus(X, E, R) :- sinus(X / 3, E, R1), R = 3 \* R1 - 4 \* R1 ^ 3.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте точность вычисления: "),

E = stdio::read(), hasDomain(real, E),

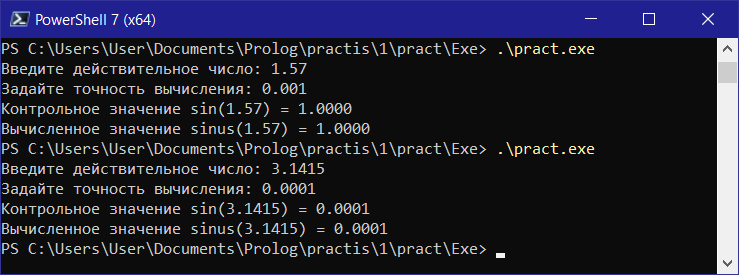
stdio::writef("Контрольное значение sin(%) = %.4f \n", X, math::sin(X)),

sinus(X, E, Res),

stdio::writef("Вычисленное значение sinus(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем синус 90 и 180 градусов, т. е. π/2 и π (в радианах примерно 1,57 и 3,1415). Получаем значения примерно равные 1 и 0 соответственно.



1. Предикат, вычисляющий функцию sin(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции sin(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Для вычисления функции синус определим предикат sinusR(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить синус, 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

factor(0, 1) :- !.

factor(N, F) :- factor(N - 1, F1), F = N \* F1.

sinusR(X, 1, X) :- !.

sinusR(X, N, R) :- N1 = N - 1, sinusR(X, N1, R1), factor(2 \* N - 1, F),

R = (-1) ^ (N - 1) \* X ^ (2 \* N - 1) / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

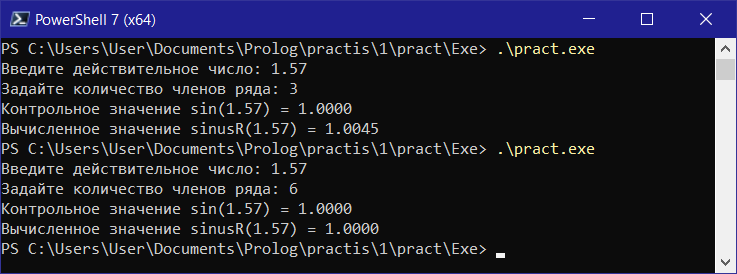
stdio::writef("Контрольное значение sin(%) = %.4f \n", X, math::sin(X)),

sinusR(X, N, Res),

stdio::writef("Вычисленное значение sinusR(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем синус 90 градусов, т. е. π/2 (в радианах примерно 1,57) и с вычислением 3 членов ряда. Полученное значение отличается от контрольного на 0,0045. При увеличении количества вычисленных членов ряда до 6 вычисленное значение синуса совпадает с контрольным с точностью до 4 знаков после запятой.

1. Предикат, вычисляющий по натуральному числу N сумму четных чисел, не превосходящих N.

В качестве условия остановки рекурсии примем факт, что 0 является четным числом, соответственно, минимальная сумма четных чисел при *N =* 0 равна 0. Таким образом, рекурсивно получаем, что сумма четных чисел, не превосходящихN, вычисляется как *SN = SN-1 + N* при четном N и как *SN = SN-1* при нечетном. Для вычисления суммы четных чисел, не превосходящихN, определим предикат myEvenSum(integer, integer [out]), принимающий 1-ый аргумент как натуральное числоN, а 2-ой является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:)

myEvenSum(0, 0) :- !.

myEvenSum(N, R) :-

myEvenSum(N - 1, R1),

if N mod 2 = 0 then R = R1 + N

else R = R1

end if.

run() :-

stdio::write("Введите натурально число: "),

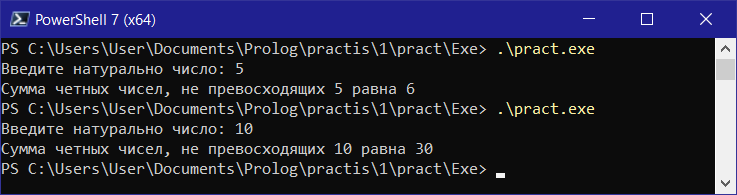
N = stdio::read(), hasDomain(integer, N),

myEvenSum(N, Res),

stdio::writef("Сумма четных чисел, не превосходящих % равна %", N, Res),

\_ = stdio::readLine().

При тестировании предиката проверяем числа 5 и 10. Соответственно, при *N = 5 S5 = 0 + 2 + 4 = 6*, а при *N = 10 S10 = 0 + 2 + 4 + 6 + 8 + 10 = 30*.

1. Предикат, вычисляющий функцию exp(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции exp(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Для вычисления функции exp(x) определим предикат expa(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить функцию exp(x), 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

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

expa(X, N, R) :- N1 = N - 1, expa(X, N1, R1),

factor(N, F), R = X ^ N / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

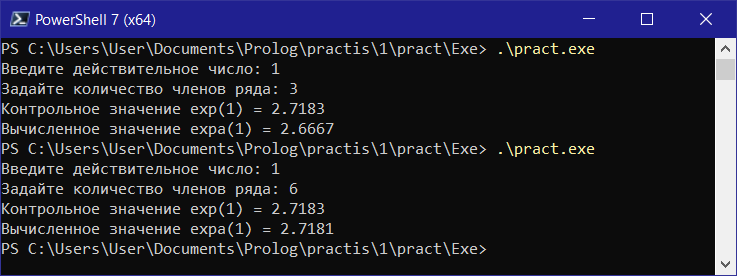
stdio::writef("Контрольное значение exp(%) = %.4f \n", X, math::exp(X)),

expa(X, N, Res),

stdio::writef("Вычисленное значение expa(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем экспоненту единицы, т. е. . При вычислении 3 членов ряда вычисленное значение экспоненты отличается от контрольного на 0,0516. При увеличении количества вычисленных членов ряда до 6 вычисленное значение экспоненты отличается от контрольного на 0,0002.

1. Предикат, вычисляющий функцию exp (x), используя разложение в ряд по заданному количеству членов ряда.

Т.к. данное задание совпадает с предыдущим, предположим, что в задании допущена описка и необходимо вычислить exp-1(x), т.е .

Для вычисления функции exp-1(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Для вычисления функции exp-1(x) определим предикат expaT(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить функцию exp-1(x), 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

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

expaT(X, N, R) :- N1 = N - 1, expaT(X, N1, R1),

factor(N, F), R = (-1) ^ N \* X ^ N / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

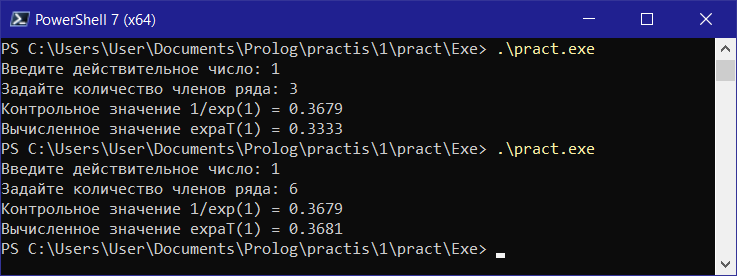
stdio::writef("Контрольное значение 1/exp(%)=%.4f \n", X, 1/math::exp(X)),

expaT(X, N, Res),

stdio::writef("Вычисленное значение expaT(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем exp-1(1), т. е. . При вычислении 3 членов ряда вычисленное значение отличается от контрольного на 0,0346. При увеличении количества вычисленных членов ряда до 6 вычисленное значение экспоненты отличается от контрольного на 0,0002.

1. Предикат, вычисляющий функцию cos(x) с заданной точностью.

Для вычисления функции cos(x) с заданной точностью используем формулу косинус тройного угла , откуда . В качестве условия остановки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,0001. Для вычисления функции cos(x) с заданной точностью определим предикат myCos(real, real,   
real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить косинус, 2-ой — заданная точность,   
а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

myCos(X, E, X) :- X < E, !.

myCos(X, E, R) :-

myCos(X / 3, E, R1),

R = 4 \* R1 ^ 3 - 3 \* R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте точность вычисления: "),

E = stdio::read(), hasDomain(real, E),

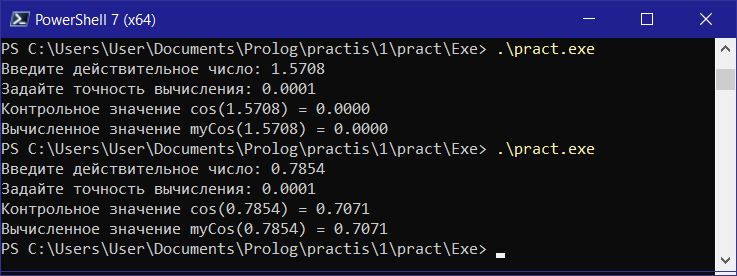
stdio::writef("Контрольное значение cos(%) = %.4f \n", X, math::cos(X)),

myCos(X, E, Res),

stdio::writef("Вычисленное значение myCos(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем косинус 45 и 90 градусов, т. е. π/4 и π/2 (в радианах примерно 0,7854 и 1,5708), с точностью 0,0001. Вычисленные и контрольные значения совпадают.

1. Предикат, вычисляющий функцию cos(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции cos(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Для вычисления функции синус определим предикат myCosR(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить косинус, 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

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

myCosR(X, N, R) :-

N1 = N - 1,

myCosR(X, N1, R1),

factor(2 \* N, F),

R = (-1) ^ N \* X ^ (2 \* N) / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

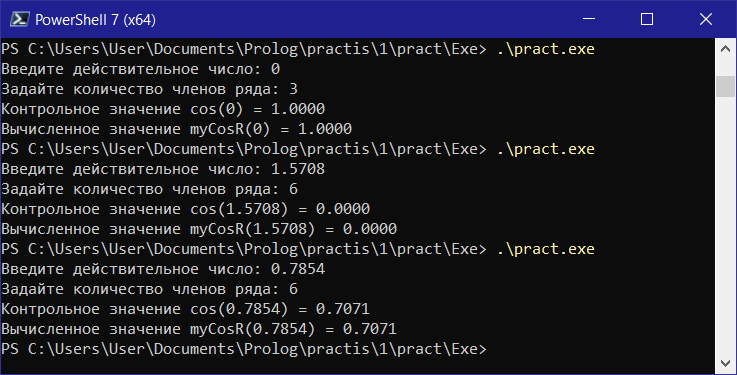
stdio::writef("Контрольное значение cos(%) = %.4f \n", X, math::cos(X)),

myCosR(X, N, Res),

stdio::writef("Вычисленное значение myCosR(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем косинус 0, 90 и 45 градусов, т. е. 0, π/2 и π/4 (в радианах примерно 0, 1,57 и 0,7854) и с вычислением 6 членов ряда. При вычислении 6 членов ряда вычисленное значение косинуса совпадает с контрольным с точностью до 4 знаков после запятой.

1. Предикат, вычисляющий рекурсивно значение , используя функцию квадратного корня.

В качестве условия остановки рекурсии примем условие, что при конечный результат равен квадратному корню из *х*. Для вычисления функции определим предикат nroot(integer, math::uReal, math::uReal [out]), принимающий 1-ый аргумент в качестве целого числа *K*, 2-ой — в качестве *х*, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

nroot(1, X, R) :- R = math::sqrt(X), !.

nroot(K, X, R) :- K1 = K div 2,

nroot(K1, X, R1), R = math::sqrt(R1).

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(math::uReal, X),

stdio::write("Введите значение K/2: "),

K = stdio::read(), hasDomain(integer, K),

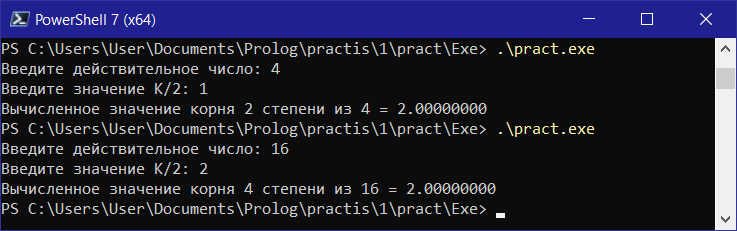
nroot(K, X, Res),

stdio::writef("Вычисленное значение корня % степени из % = %.8f",

K \* 2, X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем квадратный корень из 4 и корень 4-ой степени из 16.



1. Предикат, вычисляющий по натуральному числу N произведение чисел от (N div 2) до N.

В качестве условия остановки рекурсии примем факт, что достигнуто значение N div 2. Определим дополнительный предикат multiDiv2(unsigned, unsigned, unsigned [out]), принимающий 1-ый аргумент как натуральное числоN, 2-ой промежуточное значение, а 3-ий является выходным аргументом. Для вычисления произведение чисел от (N div 2) до N определим предикат myMult(unsigned, unsigned [out]), принимающий 1-ый аргумент как натуральное числоN, а 2-ой является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:)

multDiv2(N, M, M) :- M = N div 2, !.

multDiv2(N, M, R) :- M1 = M - 1, multDiv2(N, M1, R1), R = R1 \* M.

myMult(N, R) :- multDiv2(N, N, R).

run() :-

stdio::write("Введите натуральное число: "),

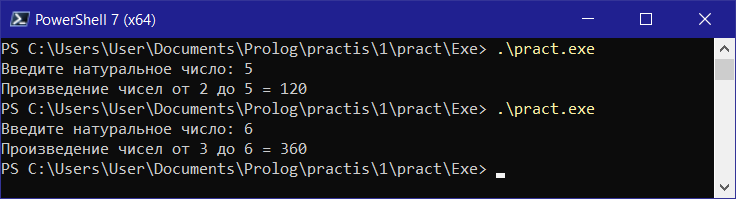
N = stdio::read(), hasDomain(unsigned, N),

myMult(N, Res),

stdio::writef("Произведение чисел от % до % = %", N div 2, N, Res),

\_ = stdio::readLine().

При тестировании предиката проверяем числа 5 и 6. Соответственно, при *N = 5 П5 = 2 × 3 × 4 × 5 = 120*, а при *N = 6 П6 = 3 × 4 × 5 × 6 = 360*.

1. Предикат, вычисляющий по натуральному числу N среднеарифметическое натуральных четных чисел, не превосходящих N.

Для вычисления среднеарифметического натуральных четных чисел используем предикат myEvenSum(integer, integer [out]) из 9-го задания, вычисляющий сумму четных чисел, не превосходящих N. Для вычисления среднеарифметического полученный результат суммы четных чисел разделим на N div 2. Для этого определим предикат evenAverage(integer, real [out]), принимающий 1-ый аргумент как натуральное числоN, а 2-ой является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:)

evenAverage(N, R) :- myEvenSum(N, R1), R = R1 / (N div 2).

run() :-

stdio::write("Введите натуральное число: "),

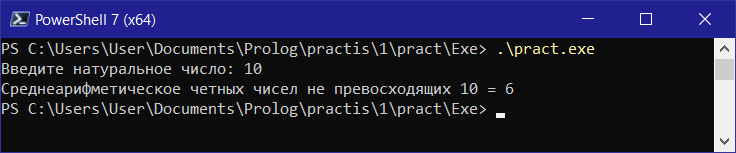
N = stdio::read(), hasDomain(integer, N),

evenAverage(N, Res),

stdio::writef("Среднеарифметическое четных чисел не превосходящих % = %", N, Res),

\_ = stdio::readLine().

При тестировании предиката проверяем число 10. Соответственно, при *N = 10*  среднеарифметическое четных чиселравно (*2 + 4 + 6 + 8 + 10) / 5 = 6*.

1. Предикат, вычисляющий среднегеометрическое натуральных чисел, кратных 3, не превосходящих N.

Для вычисления среднегеометрического натуральных чисел, кратных 3, не превосходящих N, используем дополнительные предикаты:   
prod : (real, real, real [out]) для вычисления произведения чисел, кратных 3; pow : (math::uReal, real, real [out]) для вычисления степени (в том числе и меньшей 1); count : (real, real, real [out]) для вычисления суммы чисел, кратных 3. Для вычисления среднегеометрического определим предикат avGeo3(integer, real [out]), принимающий 1-ый аргумент как натуральное числоN, а 2-ой является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:)

prod(N, N, N) :- !.

prod(N, K, 1) :- K > N, !.

prod(N, K, R) :- K3 = K + 3, prod(N, K3, U), R = U \* K.

pow(X, Y, R) :- math::exp(Y \* math::ln(X)) = R.

count(N, K, 1) :- K = N, !.

count(N, K, 0) :- K > N, !.

count(N, K, R) :- K3 = K + 3, count(N, K3, R1), R = R1 + 1.

avGeo3(N, R) :- count(N, 3, P), prod(N, 3, A), pow(A, 1.0 / P, R).

run() :-

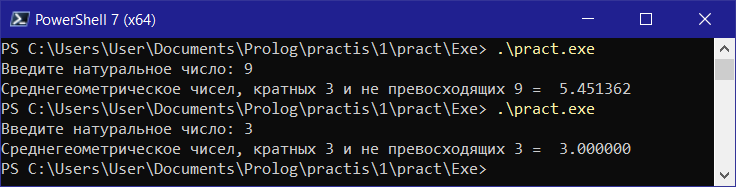
stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N),

evenAverage(N, Res),

stdio::writef("Среднеарифметическое четных чисел не превосходящих % = %", N, Res),

\_ = stdio::readLine().

При тестировании предиката проверяем числа 9 и 3. Соответственно, при *N = 9* среднегеометрическое чисел, кратных 3, не превосходящих 9равно , а при *N = 3* среднегеометрическое чисел, кратных 3, не превосходящих 3равно .

1. Предикат, выводящий простые числа на интервале (0, N). Использовать предикат, вычисляющий НОД.

Для вывода делителей числа используем дополнительный предикат   
myDels : (integer, integer), который перебирает числа от N до K и выводит число, если оно делит K нацело. Для вывода всех делителей числа определим предикат outDels : (integer), принимающий аргумент как натуральное число. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

myDels(N, K) :- N > K, !.

myDels(N, K) :-

if K = 1 or K = 2 or K = 3

or K = 5 or K = 7 or K = 11 or K = 13 or K = 17

or K = 19 or K = 23 or K = 29 or K = 31 or K = 37

or K = 41 or K = 43 or K = 47 or K = 53 or K = 59

or K = 61 or K = 67 or K = 71 or K = 73 or K = 79

or K = 83 or K = 89 or K = 97 or K = 101

then

stdio::writef("% ", K)

end if, myDels(N + 1, K).

outDels(N) :- myDels(1, N).

run() :-

stdio::write("Введите натуральное число: "),

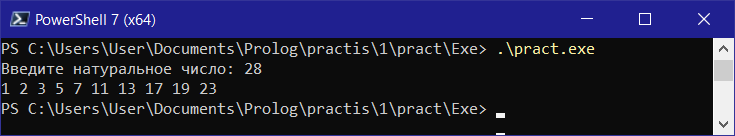
N = stdio::read(), hasDomain(integer, N),

stdio::writef("Все делители числа %:\n", N),

outDels(N),

\_ = stdio::readLine().

При тестировании предиката проверяем число 28. Соответственно, при *N = 28* будут выведены числа 1, 2, 3, 5, 7, 11, 13, 17, 19, 23.

1. Предикат, выводящий все делители числа N.

Для вывода делителей числа используем дополнительный предикат   
myDels : (integer, integer), который перебирает числа от N до K и выводит число, если оно делит K нацело. Для вывода всех делителей числа определим предикат outDels : (integer), принимающий аргумент как натуральное число. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

myDels(N, K) :- N > K, !.

myDels(N, K) :-

if K mod N = 0 then

stdio::writef("% ", N)

end if, myDels(N + 1, K).

outDels(N) :- myDels(1, N).

run() :-

stdio::write("Введите натуральное число: "),

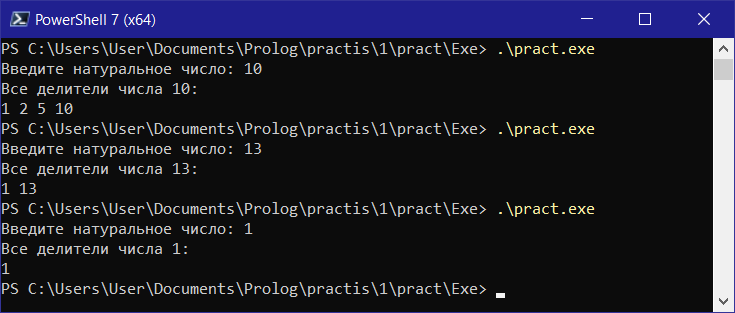
N = stdio::read(), hasDomain(integer, N),

stdio::writef("Все делители числа %:\n", N),

outDels(N),

\_ = stdio::readLine().

При тестировании предиката проверяем числа 10, 13 и 1. Соответственно, при *N = 10* делителями будут 1, 2, 5 и 10; при *N = 13* делителями будут только 1 и 13, т. к. 13 является простым числом. У единицы один делитель: это она сама.

1. Предикат, вычисляющий функцию tg(x) с заданной точностью.

Для вычисления функции tg(x) с заданной точностью используем формулу тангенс тройного угла , откуда . В качестве условия остановки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,0001. Для вычисления функции tg(x) с заданной точностью определим предикат myTan(real, real,   
real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить тангенс, 2-ой — заданная точность,   
а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

myTan(X, E, X) :- X < E, !.

myTan(X, E, R) :- myTan(X / 3, E, R1),

R = (3 \* R1 - R1 ^ 3) / (1 - 3 \* R1 ^ 2).

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте точность вычисления: "),

E = stdio::read(), hasDomain(real, E),

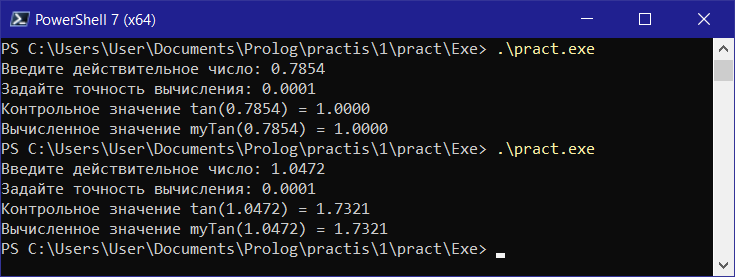
stdio::writef("Контрольное значение tan(%) = %.4f \n", X, math::tan(X)),

myTan(X, E, Res),

stdio::writef("Вычисленное значение myTan(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем тангенс 45 и 60 градусов, т. е. π/4 и π/3 (в радианах примерно 0,7854 и 1,0472), с точностью 0,0001. Вычисленные и контрольные значения совпадают.

1. Предикат, вычисляющий функцию tg(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции tg(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Так же в формуле используется число Бернулли *B2n*, которое зададим магическим способом для n не больших 10!! Для вычисления тангенса определим предикат myTanR(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить тангенс, 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

factor(0, 1) :- !.

factor(N, F) :- factor(N - 1, F1), F = N \* F1.

myTanR(X, 1, X) :- !.

myTanR(X, N, R) :- N1 = N - 1, myTanR(X, N1, R1),

factor(2 \* N, F),

if N = 10 then B = -174611 / 330

elseif N = 9 then B = 43867 / 798

elseif N = 8 then B = -3617 / 510

elseif N = 7 then B = 7 / 6

elseif N = 6 then B = -691 / 2730

elseif N = 5 then B = 5 / 66

elseif N = 4 then B = -1 / 30

elseif N = 3 then B = 1 / 42

elseif N = 2 then B = -1 / 30

elseif N = 1 then B = 1 / 6

else B = 1 end if,

R = (-1) ^ (N + 1) \* 2 ^ (2 \* N) \* (2 ^ (2 \* N) - 1) \* B \* X ^ (2 \* N – 1)

/ F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

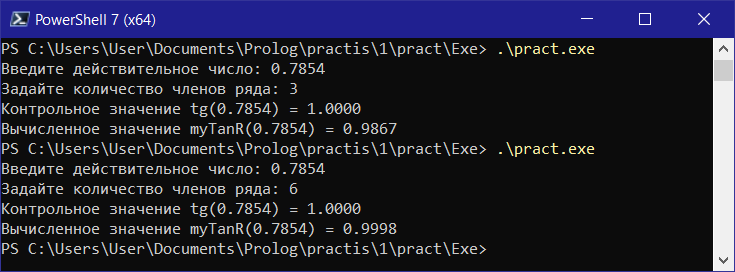
stdio::writef("Контрольное значение tg(%) = %.4f \n", X, math::tan(X)),

myTanR(X, N, Res), stdio::writef("Вычисленное значение myTanR(%) = %.4f",

X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем тангенс 45 градусов, т. е. π/4 (в радианах примерно 0,7854) и с вычислением 3 членов ряда. Полученное значение отличается от контрольного на 0,0133. При увеличении количества вычисленных членов ряда до 6 вычисленное значение тангенса отличается от контрольного на 0,0002.

1. Предикат, вычисляющий по натуральному числу N произведение чисел, кратных 3 не превосходящих N.

Для вычисления произведения чисел, кратных 3, не превосходящих N, используем предикат prod : (real, real, real [out]) из 17-го задания. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

prod(N, N, N) :- !.

prod(N, K, 1) :- K > N, !.

prod(N, K, R) :- K3 = K + 3, prod(N, K3, U), R = U \* K.

run() :-

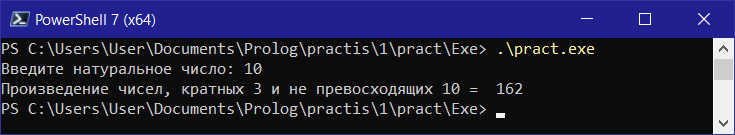
stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N), prod(N, 3, Res),

stdio::writef("Произведение чисел, кратных 3 и не превосходящих % = %",

N, Res),

\_ = stdio::readLine().

При тестировании предиката проверяем число 10. Соответственно, при *N=**10* произведение чисел, кратных 3, не превосходящих 10равно .

1. Предикат, вычисляющий функцию ln(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции ln(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу , причем ряд сходится абсолютно на интервале (–1, 1). В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Для вычисления натурального логарифма определим предикат myLnR(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа + 1, для которого необходимо вычислить логарифм, 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

factor(0, 1) :- !.

factor(N, F) :- factor(N - 1, F1), F = N \* F1.

myLnR(X, 1, X) :- !.

myLnR(X, N, R) :- N1 = N - 1, myLnR(X, N1, R1),

factor(N, F), R = (-1) ^ (N + 1) \* X ^ N / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

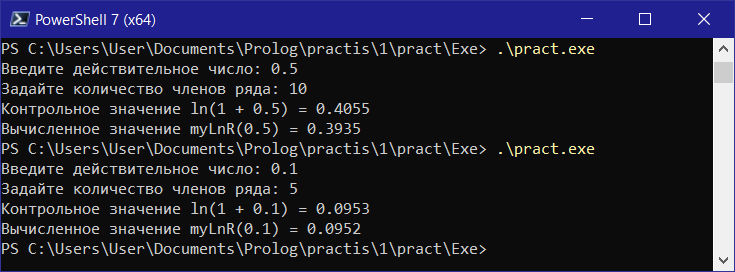
stdio::writef("Контрольное значение ln(1 + %) = %.4f \n", X, math::ln(1 + X)),

myLnR(X, N, Res),

stdio::writef("Вычисленное значение myLnR(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем логарифм 1.5, т. е. 1 + 0,5, с вычислением 10 членов ряда. Полученное значение отличается от контрольного на 0,012.

При тестировании предиката вычисляем логарифм 1.1, т. е. 1 + 0,1, с вычислением 5 членов ряда. Полученное значение отличается от контрольного на 0,0001. Как видно, при приближении к 1 точность увеличивается независимо от количества вычисленных членов ряда.

1. Предикат, вычисляющий функцию ln(x), используя разложение в ряд по заданному количеству членов ряда.

Повтор 23-го задания.

1. Предикат, вычисляющий функцию ctg(x) с заданной точностью.

Для вычисления функции ctg(x) с заданной точностью используем формулу котангенс тройного угла , откуда . В качестве условия остановки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,0001. Для вычисления функции ctg(x) с заданной точностью определим предикат myCotan(real, real,   
real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить котангенс, 2-ой — заданная точность,   
а 3-ий является выходным аргументом. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

myCotan(X, E, X) :- X < E, !.

myCotan(X, E, R) :- myCotan(X / 3, E, R1),

R = (3 \* R1 - R1 ^ 3) / (1 - 3 \* R1 ^ 2).

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте точность вычисления: "),

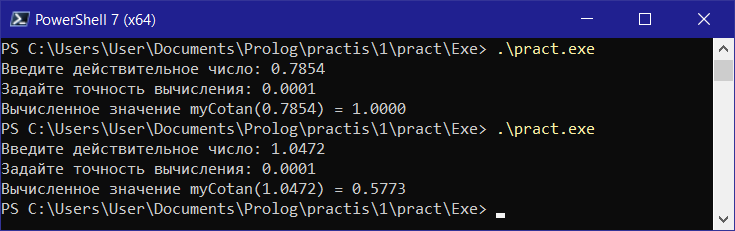
E = stdio::read(), hasDomain(real, E),

myCotan(X, E, Res),

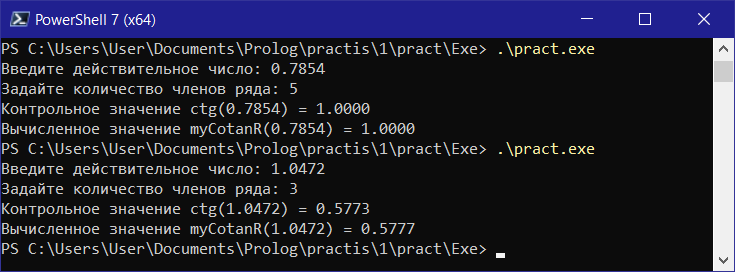
stdio::writef("Вычисленное значение myCotan(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем котангенс 45 и 60 градусов, т. е. π/4 и π/3 (в радианах примерно 0,7854 и 1,0472), с точностью 0,0001. Вычисленные и контрольные значения совпадают.

1. Предикат, вычисляющий функцию ctg(x), используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции ctg(x) с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу . В качестве условия остановки рекурсии примем условие, что при *,* согласно формуле разложения,. Дополнительно, для вычисления используем предикат вычисления факториала числа factor(integer, integer [out]), принимающий аргумент как целое число и возвращающий результат вторым аргументом. Так же в формуле используется число Бернулли *B2n*, которое зададим магическим способом для n не больших 10!! Для вычисления котангенса определим предикат myCotanR(real, integer, real [out]), принимающий 1-ый аргумент в качестве действительного числа, для которого необходимо вычислить котангенс, 2-ой — заданное количество членов ряда, а 3-ий является выходным аргументом. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

При тестировании предиката вычисляем котангенс 45 градусов, т. е. π/4 (в радианах примерно 0,7854), с вычислением 5 членов ряда. Полученное значение не отличается от контрольного.

factor(0, 1) :- !.

factor(N, F) :- factor(N - 1, F1), F = N \* F1.

myCotanR(X, 0, 1 / X) :- !.

myCotanR(X, N, R) :- N1 = N - 1, myCotanR(X, N1, R1),

factor(2 \* N, F),

if N = 10 then B = -174611 / 330

elseif N = 9 then B = 43867 / 798

elseif N = 8 then B = -3617 / 510

elseif N = 7 then B = 7 / 6

elseif N = 6 then B = -691 / 2730

elseif N = 5 then B = 5 / 66

elseif N = 4 then B = -1 / 30

elseif N = 3 then B = 1 / 42

elseif N = 2 then B = -1 / 30

elseif N = 1 then B = 1 / 6

else B = 1 end if,

R = (-1) ^ N \* 2 ^ (2 \* N) \* B \* X ^ (2 \* N - 1) / F + R1.

run() :-

stdio::write("Введите действительное число: "),

X = stdio::read(), hasDomain(real, X),

stdio::write("Задайте количество членов ряда: "),

N = stdio::read(), hasDomain(integer, N),

stdio::writef("Контрольное значение ctg(%) = %.4f \n", X,

math::cos(X) / math::sin(X)),

myCotanR(X, N, Res),

stdio::writef("Вычисленное значение myCotanR(%) = %.4f", X, Res),

\_ = stdio::readLine().

При тестировании предиката вычисляем котангенс 60 градусов, т. е. π/3 (в радианах примерно 1,0472), с вычислением 3 членов ряда. Полученное значение отличается от контрольного на 0,0004.

**Лабораторная работа № 2.**

**Название работы:** Списки. Сортировка списков.

**Цели работы:** Изучить структуру создания списков в языке Турбо Пролог.

**Задание:**

1. Создайте предикат, заменяющий в исходном списке первое вхождение заданного значения другим.
2. Создайте предикат, заменяющий в исходном списке все вхождения заданного значения другим.
3. Создайте предикат, порождающий по заданному натуральному числу N список, состоящий из натуральных чисел от 1 до N (по возрастанию).
4. Создайте предикат, порождающий по заданному натуральному числу N список, состоящий из натуральных чисел от N до 1 (по убыванию).
5. Создайте предикат, порождающий по заданному натуральному числу N список, состоящий из N случайных натуральных чисел из промежутка от 1 до 100.
6. Создайте предикат, порождающий по заданным числам N, M, K список, состоящий из N случайных натуральных чисел из промежутка от M до K.
7. Создайте предикат, порождающий по заданным числам M, K список, состоящий из случайного количества случайных чисел из промежутка от M до K.
8. Создайте предикат, который увеличивает элементы исходного списка на величину K.
9. Создайте предикат, переводящий список цифр от 0 до 9 в список соответствующих им названий (строк).
10. Создайте предикат, переводящий список цифр от 0 до 9 в список соответствующих им римских чисел.
11. Создайте предикат, переводящий список римских чисел в список соответствующих им арабских чисел (диапазон от 1 до 20).
12. Создайте предикат, удваивающий значения элементов списка.
13. Создайте предикат, преобразующий список, элементами которого являются числа, в список, элементы которого неотрицательны.
14. Создайте предикат, преобразующий исходный список в список позиций отрицательных элементов.
15. Создайте предикат, удаляющий из исходного списка элементы с четными номерами.
16. Создайте предикат, который разделит исходный список из целых чисел на два списка: список положительных чисел и список отрицательных чисел.
17. Создайте предикат, разделяющий исходный список на два подсписка. В первый из них должны попасть элементы с нечетными номерами, во второй – элементы с четными номерами.
18. Создайте предикат, вычисляющий по списку и числу, подсписок исходного списка, начинающийся с элемента с указанным номером.
19. Создайте предикат, осуществляющий удаление указанного количества последних элементов исходного списка.
20. Создайте предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов из начала списка, во второй – оставшиеся элементы.
21. Создайте предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов с конца списка, во второй – оставшиеся элементы.
22. Создайте предикат, находящий предпоследний элемент списка.
23. Создайте предикат, удаляющий предпоследний элемент списка.
24. Создайте предикат, заменяющий в исходном списке два подряд идущих одинаковых элемента одним.
25. Создайте предикат, удаляющий в исходном списке все повторные вхождения элементов.
26. Создайте предикат, осуществляющий перестановку двух элементов списка с заданными номерами.
27. Создайте предикат, генерирующий все перестановки элементов списка, указанного в качестве первого аргумента предиката.
28. Создайте предикат, осуществляющий циклический сдвиг элементов списка на один влево (вправо).
29. Создайте предикат, осуществляющий циклический сдвиг элементов списка на заданное количество шагов влево (вправо).
30. Создайте предикат, осуществляющий поэлементное перемножение соответствующих элементов двух исходных списков.
31. Создайте предикат, вычисляющий скалярное произведение векторов, заданных списками целых чисел.
32. Создайте предикат, осуществляющий подсчет числа вхождений каждого элемента исходного списка. Ответом должен быть список пар, в которых первая компонента – элемент исходного списка, вторая – число его вхождений в первоначальный список.
33. Создайте предикат, определяющий первую позицию подсписка в списке.
34. Создайте предикат, добавляющий элементы одного списка во второй список, начиная с заданной позиции.
35. Создайте предикат, возвращающий по списку и двум числам M и N подсписок исходного списка, состоящий из элементов с номерами от M до N.
36. Создайте предикат, формирующий список простых чисел, не превосходящих данного числа.
37. Создайте предикат, транспонирующий матрицу, заданную списком списков.

**Выполнение практической работы.**

Изучены теоретические сведения практической работы о создании списков на языке Пролог.

Для работы со списками объявим списочный домен списка целых чисел intList:

domains

intList = integer\*.

srtList = string\_list.

1. Предикат, заменяющий в исходном списке первое вхождение заданного значения другим.

Определим предикат replaceFirst : (integer, integer, intList, intList [out]), принимающий 1-ый аргумент как искомое число, 2-ой аргумент как число для замены искомого, 3-ий аргумент — исходный список, а 4-ый является выходным аргументом и представляет список с замененным искомым значением. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

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

replaceFirst(F, N, [F | T1], [N | T1]) :- !.

replaceFirst(F, N, [Y | T1], [Y | T2]) :- replaceFirst(F, N, T1, T2).

run() :-

FL = [1, 2, 3, 4, 5],

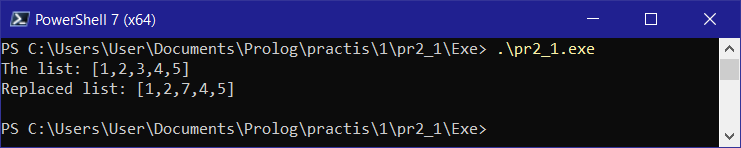
stdio::writef("The list: %\n", FL),

replaceFirst(3, 7, FL, L),

stdio::writef("Replaced list: %\n", L),

\_ = stdio::readLine().

При тестировании предиката заменяем в исходном списке [1, 2, 3, 4, 5] значение 3 на значение 7: получаем новый список [1, 2, 7, 4, 5].

1. Предикат, заменяющий в исходном списке все вхождения заданного значения другим.

Определим предикат replaceAll : (integer, integer, intList, intList [out]), принимающий 1-ый аргумент как искомое число, 2-ой аргумент как число для замены искомого, 3-ий аргумент — исходный список, а 4-ый является выходным аргументом и представляет список с замененными искомыми значениями. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

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

replaceAll(Old, New, [Old | Tail1], [New | Tail2]) :- !,

replaceAll(Old, New, Tail1, Tail2).

replaceAll(Old, New, [Y | Tail1], [Y | Tail2]) :-

replaceAll(Old, New, Tail1, Tail2).

run() :-

FL = [1, 2, 3, 4, 3],

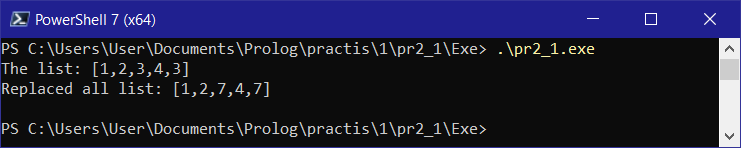
stdio::writef("The list: %\n", FL),

replaceAll(3, 7, FL, L2),

stdio::writef("Replaced all list: %\n", L2),

\_ = stdio::readLine().

При тестировании предиката заменяем в исходном списке [1, 2, 3, 4, 3] значение 3 на значение 7: получаем новый список [1, 2, 7, 4, 7].

1. Предикат, порождающий по заданному натуральному числу N список, состоящий из натуральных чисел от 1 до N (по возрастанию).

Определим предикат genUpFromTo : (integer, integer, intList [out]), принимающий 1-ый аргумент как первый элемент генерируемого списка, 2-ой аргумент как последний элемент генерируемого списка, а 3-ий является выходным аргументом и представляет сгенерированный список от 1 до N. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

genUpFromTo(From, To, []) :- From > To, !.

genUpFromTo(From, To, [From | Tail]) :- genUpFromTo(From + 1, To, Tail).

run() :-

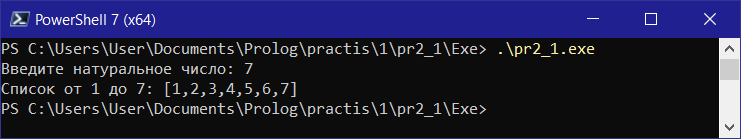
stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N),

genUpFromTo(1, N, L3), stdio::writef("Список от 1 до %: %", N, L3),

\_ = stdio::readLine().

При тестировании предиката задаем значение 7 и получаем список   
[1, 2, 3, 4, 5, 6, 7].

1. Предикат, порождающий по заданному натуральному числу N список, состоящий из натуральных чисел от N до 1 (по убыванию).

Определим предикат genDown : (integer, intList [out]), принимающий  
1-ый аргумент как число элементов генерируемого списка, а 2-ой является выходным аргументом и представляет сгенерированный список от N до 1. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

genDown(0, []) :- !.

genDown(Counter, [Counter | Tail]) :- genDown(Counter - 1, Tail).

run() :-

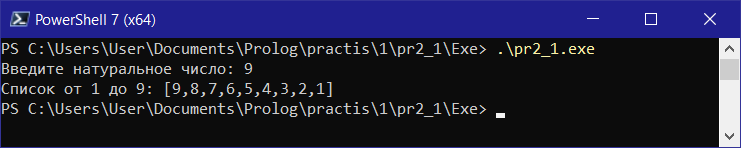
stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N),

genDown(N, L3), stdio::writef("Список от % до 1: %", N, L3),

\_ = stdio::readLine().

При тестировании предиката задаем значение 9 и получаем список   
[9, 8, 7, 6, 5, 4, 3, 2, 1].

1. Предикат, порождающий по заданному натуральному числу N список, состоящий из N случайных натуральных чисел из промежутка от 1 до 100.

Определим предикат generate5 : (integer, intList [out]), принимающий   
1-ый аргумент как количество генерируемых элементов списка, а 2-ой является выходным аргументом и представляет сгенерированный список из N случайных чисел. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

generate5(0, []) :- !.

generate5(Count, [math::random(100) | Tail]) :-

generate5(Count - 1, Tail).

run() :-

stdio::write("Введите натуральное число: "),

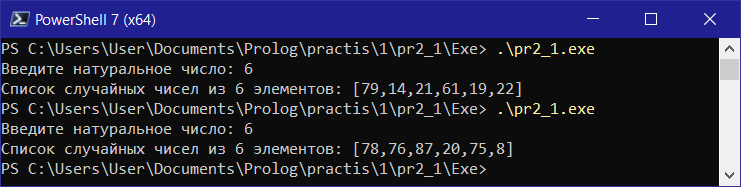
N = stdio::read(), hasDomain(integer, N),

generate5(N, L5),

stdio::writef("Список случайных чисел из % элементов: %", N, L5),

\_ = stdio::readLine().

При тестировании предиката дважды задаем значение 6 и получаем списки, состоящие из 6 случайных элементов.

1. Предикат, порождающий по заданным числам N, M, K список, состоящий из N случайных натуральных чисел из промежутка от M до K.

Определим предикат generate6 : (integer, integer, integer, intList [out]), принимающий 1-ый аргумент как количество генерируемых элементов списка, 2-ой и 3-ий являются началом и концом промежутка, из которого генерируются случайные числа, а 4-ый — выходной аргумент и представляет сгенерированный список из N случайных чисел в промежутке от M до K. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

generate6(0, \_, \_, []) :- !.

generate6(Count, RFrom, RTo, [RFrom + math::random(RTo - RFrom) | Tail]) :-

generate6(Count - 1, RFrom, RTo, Tail).

run() :-

stdio::write("Введите натуральное число: "),

N = stdio::read(), hasDomain(integer, N),

stdio::write("Введите начало промежутка: "),

M = stdio::read(), hasDomain(integer, M),

stdio::write("Введите окончание промежутка: "),

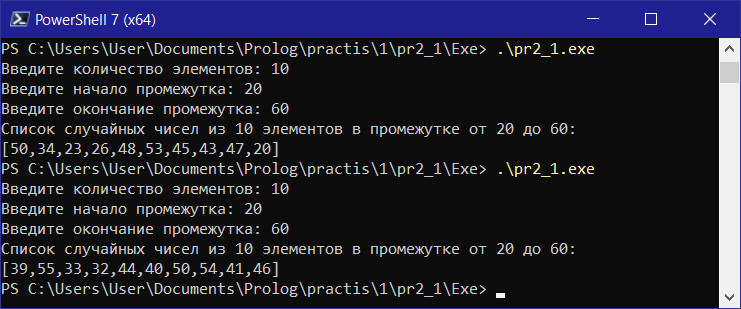
K = stdio::read(), hasDomain(integer, K),

generate6(N, M, K, L6),

stdio::writef("Список случайных чисел из % элементов в промежутке от % до %: \n%", N, M, K, L6),

\_ = stdio::readLine().

При тестировании предиката дважды задаем количество элементов равное 10, а так же промежуток от 20 до 60, и получаем списки, состоящие из 10 случайных чисел в указанном промежутке.

1. Предикат, порождающий по заданным числам M, K список, состоящий из случайного количества случайных чисел из промежутка от M до K.

Определим предикат generate7 : (integer, integer, intList [out]), принимающий 1-ый и 2-ой аргумент как начало и конец промежутка, из которого генерируются случайные числа, а 3-ий — выходной аргумент и представляет сгенерированный список из случайного количества случайных чисел из промежутка от M до K. Для это используем предикат из 6-го задания, передав ему в качестве количества элементов генерируемого списка случайное число от 0 до 20 (чтобы не загромождать вывод). Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

generate7(RFrom, RTo, List) :-

generate6(math::random(20), RFrom, RTo, List).

run() :-

stdio::write("Введите начало промежутка: "),

M = stdio::read(), hasDomain(integer, M),

stdio::write("Введите окончание промежутка: "),

K = stdio::read(), hasDomain(integer, K),

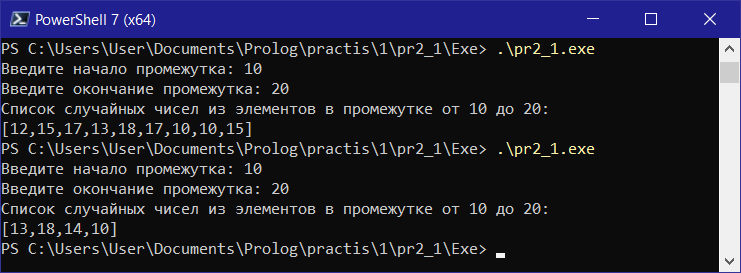
generate7(M, K, L7),

stdio::writef("Список случайных чисел в промежутке от % до %: \n%",

M, K, L7),

\_ = stdio::readLine().

При тестировании предиката дважды задаем промежуток от 10 до 20, и получаем списки, состоящие в 1-ом случае из 9, а во 2-ом - 4 случайных чисел в указанном промежутке.

1. Предикат, который увеличивает элементы исходного списка на величину K.

Определим предикат addNum : (integer, intList, intList [out]), принимающий 1-ый аргумент как значение увеличения, 2-ой — как исходный список, а 3-ий — выходной аргумент, представляющий список с увеличенными на K элементами исходного списка. Данный предикат объявляется в секции class predicates, а определяется в секции clauses, код которой приведен ниже:

addNum(K, [X | Tail], [X + K | NewTail]) :-

addNum(K, Tail, NewTail).

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

run() :-

FL = [1, 2, 3, 4, 3],

stdio::writef("Исходный список: %", FL), nl,

stdio::write("Введите значение увеличения: "),

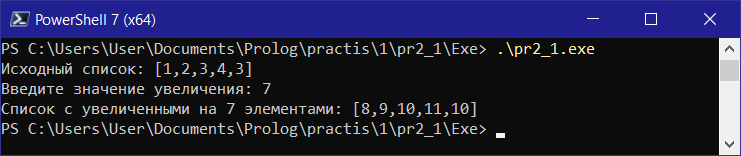
K = stdio::read(), hasDomain(integer, K),

addNum(K, FL, L8),

stdio::writef("Список с увеличенными на % элементами: %", K, L8)

\_ = stdio::readLine().

При тестировании предиката задаем значение увеличения, как 7, и получаем из исходного списка [1, 2, 3, 4, 3] список с увеличенными элементами [8, 9, 10, 11, 10].

1. Предикат, переводящий список цифр от 0 до 9 в список соответствующих им названий (строк).

Определим предикат translate2Str : (intList, strList [out]), принимающий 1-ый аргумент как список цифр, а 2-ой — выходной аргумент, представляющий список строк с наименованиями цифр. Определим дополнительный предикат d2s : (integer, string [out]), принимающий 1-ый аргумент как число, а 2-ой — выходной аргумент, являющийся строкой с наименованием цифры. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

d2s(X, S) :-

if X = 0 then S = "Ноль"

elseif X = 1 then S = "Один"

elseif X = 2 then S = "Два"

elseif X = 3 then S = "Три"

elseif X = 4 then S = "Четыре"''

elseif X = 5 then S = "Пять"

elseif X = 6 then S = "Шесть"

elseif X = 7 then S = "Семь"

elseif X = 8 then S = "Восемь"

elseif X = 9 then S = "Девять"

else S = "ХЗ"

end if.

translate2Str([], []) :- !.

translate2Str([DH | DT], [SH | Out]) :-

d2s(DH, SH), translate2Str(DT, Out).

run() :-

L = [6, 3, 0, 9, 45],

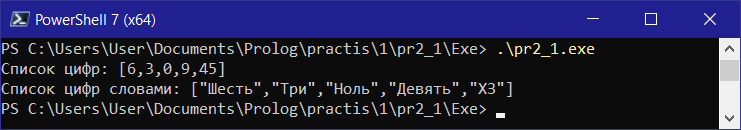
stdio::writef("Список цифр: %", L), nl,

translate2Str(L, L9),

stdio::writef("Список цифр словами: %", L9),

\_ = stdio::readLine().

При тестировании предиката задаем список чисел, из которых все, кроме последнего, можно рассмотреть как цифры: [6, 3, 0, 9, 45]; последнее число представить как цифру нельзя. В результате получаем список с наименованиями цифр и строкой «ХЗ», если число не представить как цифру: ["Шесть", "Три", "Ноль", "Девять", "ХЗ"].

1. Предикат, переводящий список цифр от 0 до 9 в список соответствующих им римских чисел.

Определим предикат translate2R : (intList, strList [out]), принимающий 1-ый аргумент как список цифр, а 2-ой — выходной аргумент, представляющий список строк с римскими цифрами. Определим дополнительный предикат d2R : (integer, string [out]), принимающий 1-ый аргумент как число, а 2-ой — выходной аргумент, являющийся строкой с римской цифрой. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

d2R(X, S) :-

if X = 10 then S = "X"

elseif X = 1 then S = "I"

elseif X = 2 then S = "II"

elseif X = 3 then S = "III"

elseif X = 4 then S = "IV"

elseif X = 5 then S = "V"

elseif X = 6 then S = "VI"

elseif X = 7 then S = "VII"

elseif X = 8 then S = "VIII"

elseif X = 9 then S = "IX"

else S = "-"

end if.

translate2R([], []) :- !.

translate2R([DH | DT], [SH | Out]) :-

d2R(DH, SH), translate2R(DT, Out).

run() :-

L = [6, 3, 1, 9, 8],

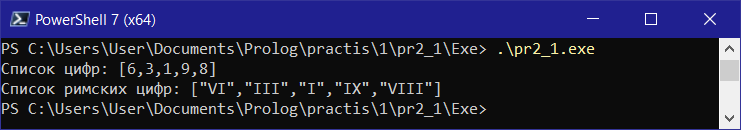
stdio::writef("Список цифр: %", L), nl,

translate2R(L, L10),

stdio::writef("Список римских цифр: %", L10),

\_ = stdio::readLine().

При тестировании предиката задаем список чисел, из которых все можно рассмотреть как цифры: [6, 3, 1, 9, 8]. В результате получаем список с римскими цифрами: ["VI", "III", "I", "IX", "VIII"].

1. Предикат, переводящий список римских чисел в список соответствующих им арабских чисел (диапазон от 1 до 20).
2. Предикат, удваивающий значения элементов списка.
3. Предикат, преобразующий список, элементами которого являются числа, в список, элементы которого неотрицательны.
4. Предикат, преобразующий исходный список в список позиций отрицательных элементов.
5. Предикат, удаляющий из исходного списка элементы с четными номерами.
6. Предикат, который разделит исходный список из целых чисел на два списка: список положительных чисел и список отрицательных чисел.
7. Предикат, разделяющий исходный список на два подсписка. В первый из них должны попасть элементы с нечетными номерами, во второй – элементы с четными номерами.
8. Предикат, вычисляющий по списку и числу подсписок исходного списка, начинающийся с элемента с указанным номером.
9. Предикат, осуществляющий удаление указанного количества последних элементов исходного списка.
10. Предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов из начала списка, во второй – оставшиеся элементы.
11. Предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов с конца списка, во второй – оставшиеся элементы.
12. Предикат, находящий предпоследний элемент списка.
13. Предикат, удаляющий предпоследний элемент списка.
14. Предикат, заменяющий в исходном списке два подряд идущих одинаковых элемента одним.
15. Предикат, удаляющий в исходном списке все повторные вхождения элементов.
16. Предикат, осуществляющий перестановку двух элементов списка с заданными номерами.
17. Предикат, генерирующий все перестановки элементов списка, указанного в качестве первого аргумента предиката.
18. Предикат, осуществляющий циклический сдвиг элементов списка на один влево (вправо).
19. Предикат, осуществляющий циклический сдвиг элементов списка на заданное количество шагов влево (вправо).
20. Предикат, осуществляющий поэлементное перемножение соответствующих элементов двух исходных списков.
21. Предикат, вычисляющий скалярное произведение векторов, заданных списками целых чисел.
22. Предикат, осуществляющий подсчет числа вхождений каждого элемента исходного списка. Ответом должен быть список пар, в которых первая компонента – элемент исходного списка, вторая – число его вхождений в первоначальный список.
23. Предикат, определяющий первую позицию подсписка в списке.
24. Предикат, добавляющий элементы одного списка во второй список, начиная с заданной позиции.
25. Предикат, возвращающий по списку и двум числам M и N подсписок исходного списка, состоящий из элементов с номерами от M до N.
26. Предикат, формирующий список простых чисел, не превосходящих данного числа.
27. Предикат, транспонирующий матрицу, заданную списком списков.