

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 $y = \sqrt[2k]{x}$, 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 $\text{tg}(x)$ с заданной точностью.
21. Создайте предикат, вычисляющий функцию $\text{tg}(x)$, 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 $\text{ctg}(x)$ с заданной точностью.
26. Создайте предикат, вычисляющий функцию $\text{ctg}(x)$, используя разложение в ряд по заданному количеству членов ряда.

Выполнение практической работы.

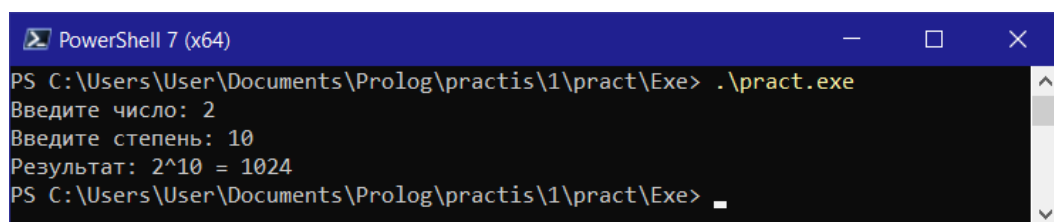
Изучены теоретические сведения практической работы о создании рекурсивных программ на языке Пролог.

1. Предикат, вычисляющий неотрицательную степень целого числа.

В качестве условия останова рекурсии примем факт, что любое целое число в нулевой степени равно 1, т. е. $P^0 = 1$. Таким образом, рекурсивно получаем, что $P^N = P^{N-1} \times P$. 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 $2^{10} = 1024$.



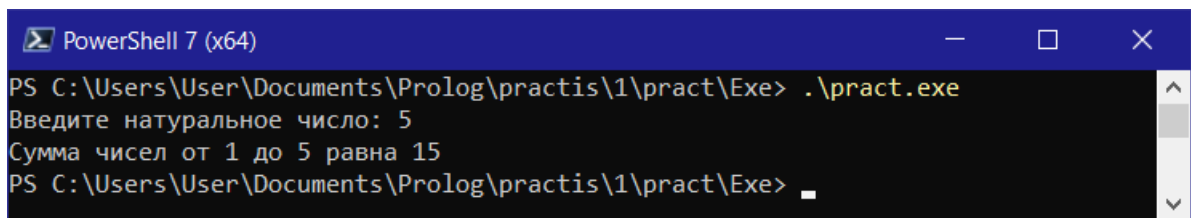
```
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe  
Введите число: 2  
Введите степень: 10  
Результат: 2^10 = 1024  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

2. 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 $S_N = S_{N-1} + N$. 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 $S_5 = 1 + 2 + 3 + 4 + 5 = 15$.



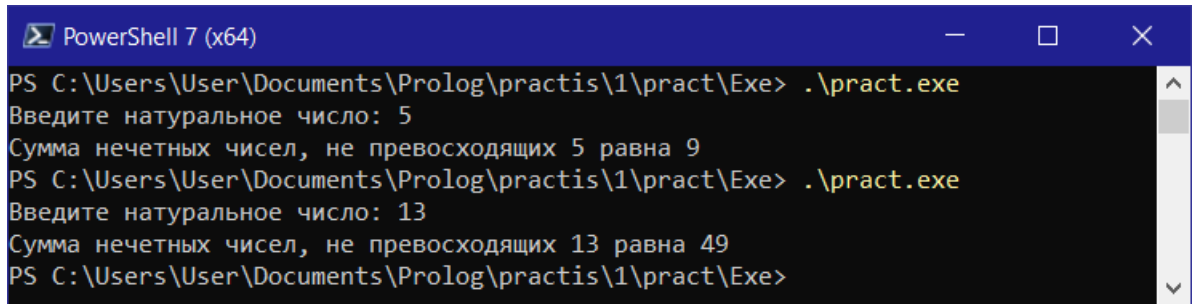
```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe  
Введите натуральное число: 5  
Сумма чисел от 1 до 5 равна 15  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

3. 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 $S_N = S_{N-1} + N$ при нечетном N и как $S_N = S_{N-1}$ 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 $N = 5$ $S_5 = 1 + 3 + 5 = 9$, а при $N = 13$ $S_{13} = 1 + 3 + 5 + 7 + 9 + 11 + 13 = 49$.



```

PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 5
Сумма нечетных чисел, не превосходящих 5 равна 9
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 13
Сумма нечетных чисел, не превосходящих 13 равна 49
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

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

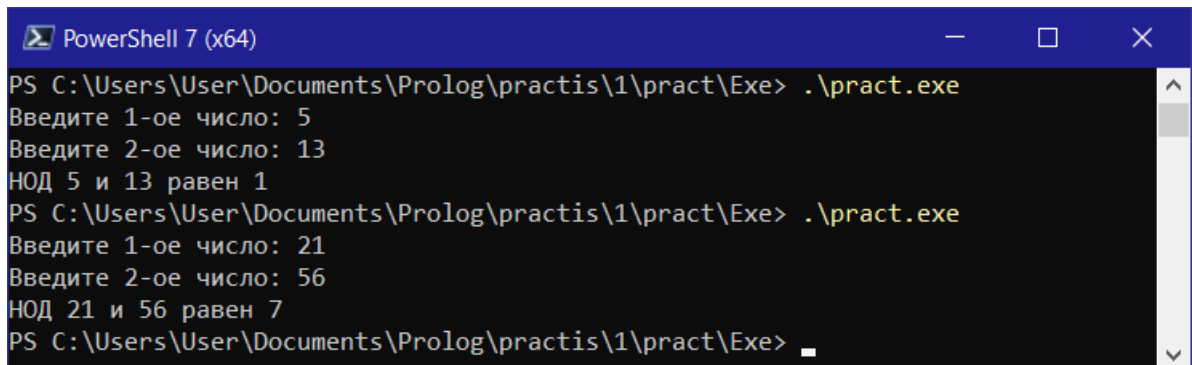
В качестве условия остановки рекурсии примем факт, что НОД натурального числа и 0 равен этому числу, т. е. $НОД(A, 0) = A$. Таким образом, рекурсивно получаем, что $НОД(A, B) = НОД(B, A \% B)$. Для вычисления НОД определим предикат `gcd(integer, integer, integer [out])`, принимающий 2 аргумента как натуральные 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.



```

PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите 1-ое число: 5
Введите 2-ое число: 13
НОД 5 и 13 равен 1
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите 1-ое число: 21
Введите 2-ое число: 56
НОД 21 и 56 равен 7
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

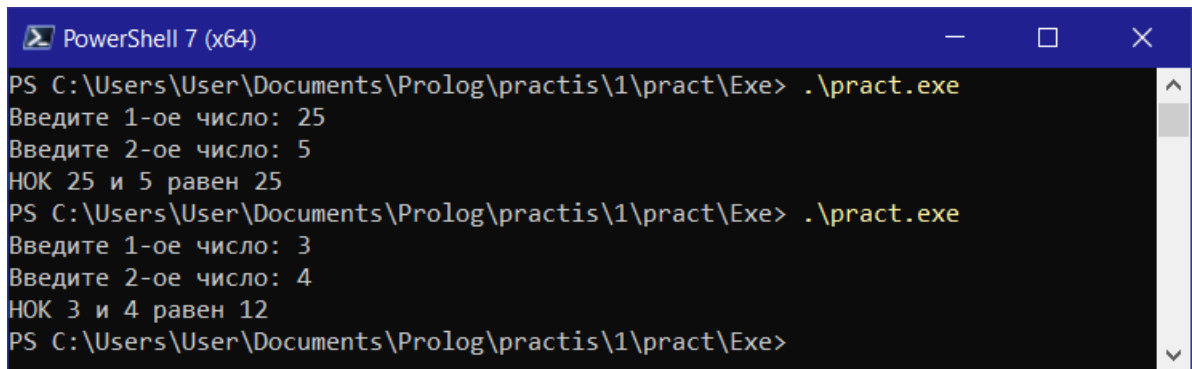
5. Предикат, вычисляющий наименьшее общее кратное двух натуральных чисел.

Для вычисления НОК двух натуральных чисел воспользуемся таким фактом, что НОК равен произведению этих чисел поделенных нацело на их НОД, т. е. $НОК(A, B) = (A \times B) / НОД(A, B)$. 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.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите 1-ое число: 25
Введите 2-ое число: 5
НОК 25 и 5 равен 25
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите 1-ое число: 3
Введите 2-ое число: 4
НОК 3 и 4 равен 12
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>
```

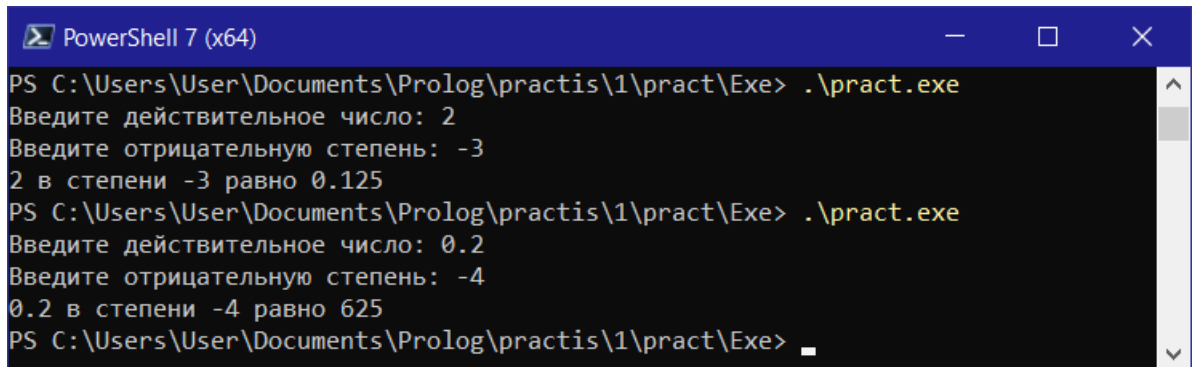
6. Предикат, вычисляющий отрицательную целую степень действительного числа.

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 $R^{-N} = \frac{1}{R^{N-1}} \cdot \frac{1}{R}$. 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 $2^{-3} = \frac{1}{2^3} = \frac{1}{8} = 0,125$ и $0,2^{-4} = \frac{1}{0,4^4} = \frac{1}{0,0256} = 625$.



```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 2
Введите отрицательную степень: -3
2 в степени -3 равно 0.125
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.2
Введите отрицательную степень: -4
0.2 в степени -4 равно 625
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

7. Предикат, вычисляющий функцию $\sin(x)$ с заданной точностью.

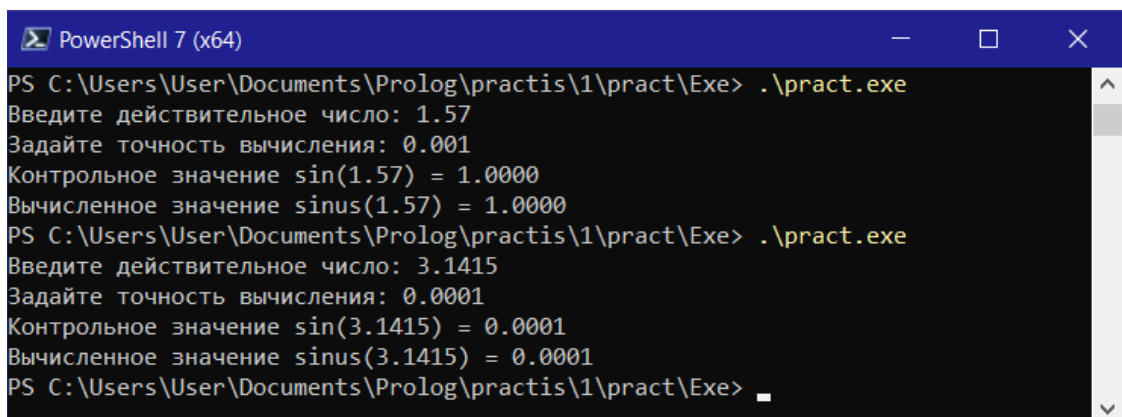
Для вычисления функции $\sin(x)$ с заданной точностью используем формулу синуса тройного угла $\sin 3x = 3 \sin x - 4 \sin^3 x$, откуда $\sin x = 3 \sin(x/3) - 4 \sin^3(x/3)$. В качестве условия останковки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, 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. $\pi/2$ и π (в радианах примерно 1,57 и 3,1415). Получаем значения примерно равные 1 и 0 соответственно.



```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.57
Задайте точность вычисления: 0.001
Контрольное значение sin(1.57) = 1.0000
Вычисленное значение sinus(1.57) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 3.1415
Задайте точность вычисления: 0.0001
Контрольное значение sin(3.1415) = 0.0001
Вычисленное значение sinus(3.1415) = 0.0001
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```


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

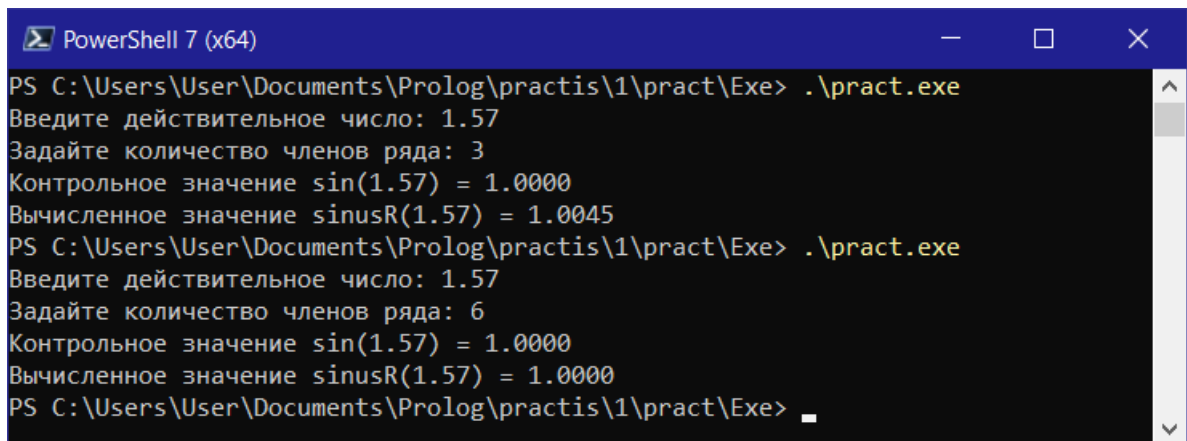
Для вычисления функции $\sin(x)$ с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу

$$\sin x = \sum_{n=1}^{\infty} \frac{(-1)^{(n-1)} \cdot x^{(2n-1)}}{(2n-1)!}.$$

В качестве условия остановки рекурсии примем условие, что при $n=1$, согласно формуле разложения, $\sin x = x$. 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. $\pi/2$ (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.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.57
Задайте количество членов ряда: 3
Контрольное значение sin(1.57) = 1.0000
Вычисленное значение sinusR(1.57) = 1.0045
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.57
Задайте количество членов ряда: 6
Контрольное значение sin(1.57) = 1.0000
Вычисленное значение sinusR(1.57) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

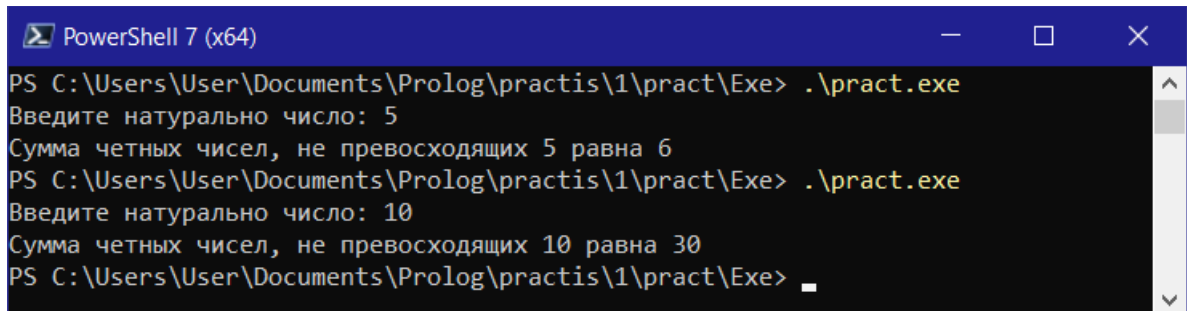
9. Предикат, вычисляющий по натуральному числу N сумму четных чисел, не превосходящих N .

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 превосходящих N , вычисляется как $S_N = S_{N-1} + N$ при четном N и как $S_N = S_{N-1}$ при нечетном. Для вычисления суммы четных чисел, не превосходящих N , определим предикат `myEvenSum(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:)

```
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 $N = 5$ $S_5 = 0 + 2 + 4 = 6$, а при $N = 10$ $S_{10} = 0 + 2 + 4 + 6 + 8 + 10 = 30$.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натурально число: 5
Сумма четных чисел, не превосходящих 5 равна 6
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натурально число: 10
Сумма четных чисел, не превосходящих 10 равна 30
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

10. 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 $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$. В качестве условия остановки рекурсии примем условие, что при $n=0$, согласно формуле разложения, $e^0 = 1$. 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. $e^1 \approx 2,7183$. 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.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1
Задайте количество членов ряда: 3
Контрольное значение exp(1) = 2.7183
Вычисленное значение expa(1) = 2.6667
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1
Задайте количество членов ряда: 6
Контрольное значение exp(1) = 2.7183
Вычисленное значение expa(1) = 2.7181
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

11. 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. $\frac{1}{e^x} = e^{-x}$.

Для вычисления функции $\exp^{-1}(x)$ с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу
$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^n}{n!}$$
. В качестве условия остановки рекурсии примем условие, что при $n=0$, согласно формуле разложения, $e^0 = 1$. 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. $e^{-1} \approx 2,7183$. 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.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1
Задайте количество членов ряда: 3
Контрольное значение 1/exp(1) = 0.3679
Вычисленное значение expaT(1) = 0.3333
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1
Задайте количество членов ряда: 6
Контрольное значение 1/exp(1) = 0.3679
Вычисленное значение expaT(1) = 0.3681
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

12. Предикат, вычисляющий функцию $\cos(x)$ с заданной точностью.

Для вычисления функции $\cos(x)$ с заданной точностью используем формулу косинус тройного угла $\cos 3x = 4 \cos^3 x - 3 \cos x$, откуда $\cos x = 4 \cos^3(x/3) - 3 \cos(x/3)$. 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. $\pi/4$ и $\pi/2$ (в радианах примерно 0,7854 и 1,5708), с точностью 0,0001. Вычисленные и контрольные значения совпадают.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.5708
Задайте точность вычисления: 0.0001
Контрольное значение cos(1.5708) = 0.0000
Вычисленное значение myCos(1.5708) = 0.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте точность вычисления: 0.0001
Контрольное значение cos(0.7854) = 0.7071
Вычисленное значение myCos(0.7854) = 0.7071
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

13. 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

$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{2n!}$. В качестве условия остановки рекурсии примем условие,

что при $n=0$, согласно формуле разложения, $\cos x = 1$. 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, $\pi/2$ и $\pi/4$ (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.

```

PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0
Задайте количество членов ряда: 3
Контрольное значение cos(0) = 1.0000
Вычисленное значение myCosR(0) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.5708
Задайте количество членов ряда: 6
Контрольное значение cos(1.5708) = 0.0000
Вычисленное значение myCosR(1.5708) = 0.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте количество членов ряда: 6
Контрольное значение cos(0.7854) = 0.7071
Вычисленное значение myCosR(0.7854) = 0.7071
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

14. Предикат, вычисляющий рекурсивно значение $y = \sqrt[2k]{x}$, используя функцию квадратного корня.

В качестве условия остановки рекурсии примем условие, что при $k=1$ конечный результат равен квадратному корню из x . Для вычисления функции $y = \sqrt[2k]{x}$ определим предикат `nroot(integer, math::uReal, math::uReal [out])`, принимающий 1-ый аргумент в качестве целого числа K , 2-ой — в качестве x , 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.

```

PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 4
Введите значение K/2: 1
Вычисленное значение корня 2 степени из 4 = 2.00000000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 16
Введите значение K/2: 2
Вычисленное значение корня 4 степени из 16 = 2.00000000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

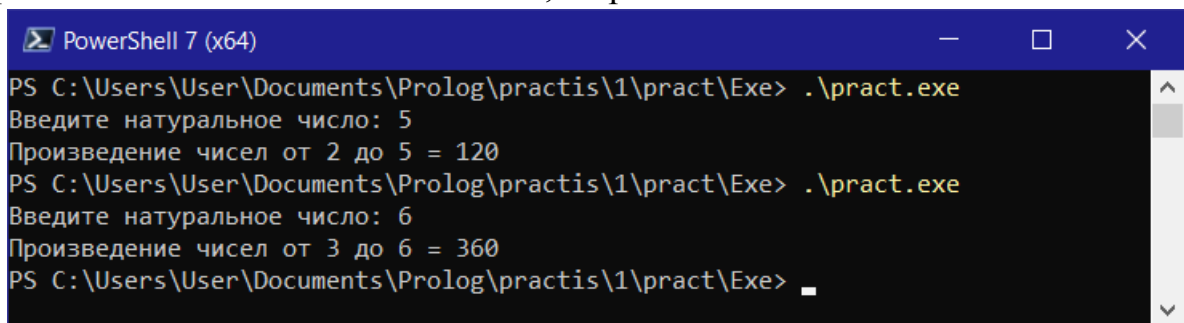
```


15. 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 число N , 2-ой промежуточное значение, а 3-ий является выходным аргументом. Для вычисления произведения чисел от $(N \div 2)$ до N определим предикат `myMult(unsigned, unsigned [out])`, принимающий 1-ый аргумент как натуральное число N , а 2-ой является выходным аргументом. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:)

```
multiDiv2(N, M, M) :- M = N div 2, !.  
multiDiv2(N, M, R) :- M1 = M - 1, multiDiv2(N, M1, R1), R = R1 * M.  
myMult(N, R) :- multiDiv2(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 $N = 5$ $P_5 = 2 \times 3 \times 4 \times 5 = 120$, а при $N = 6$ $P_6 = 3 \times 4 \times 5 \times 6 = 360$.



```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe  
Введите натуральное число: 5  
Произведение чисел от 2 до 5 = 120  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe  
Введите натуральное число: 6  
Произведение чисел от 3 до 6 = 360  
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

16. 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 число N , 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 чисел равно $(2 + 4 + 6 + 8 + 10) / 5 = 6$.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 10
Среднеарифметическое четных чисел не превосходящих 10 = 6
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

17. 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 число N, 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 9 равно $\sqrt[3]{3 \cdot 6 \cdot 9} = \sqrt[3]{162} \approx 5,451362$, а при $N = 3$ среднегеометрическое чисел, кратных 3, не превосходящих 3 равно $\sqrt[1]{3} = 3$.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 9
Среднегеометрическое чисел, кратных 3 и не превосходящих 9 = 5.451362
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 3
Среднегеометрическое чисел, кратных 3 и не превосходящих 3 = 3.000000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

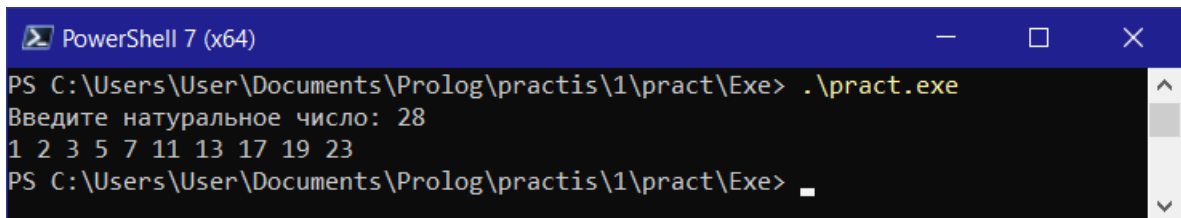
```


18. 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.



19. 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:

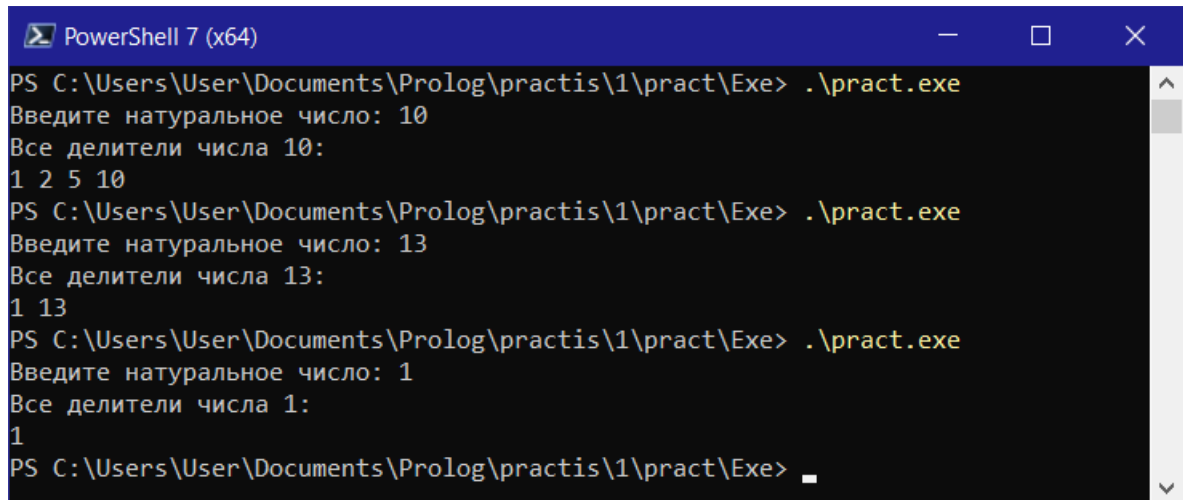
```
myDels(N, K) :- N > K, !.  
myDels(N, K) :-  
    if K mod N = 0 then  
        stdio::writef("% ", N)  
    end if, myDels(N + 1, K).  
outDels(N) :- myDels(1, N).  
run() :-  
    stdio::write("Введите натуральное число: "),  
    N = stdio::read(), hasDomain(integer, N),
```

```

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.



```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 10
Все делители числа 10:
1 2 5 10
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 13
Все делители числа 13:
1 13
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 1
Все делители числа 1:
1
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _

```

20. Предикат, вычисляющий функцию $\text{tg}(x)$ с заданной точностью.

Для вычисления функции $\text{tg}(x)$ с заданной точностью используем формулу тангенс тройного угла $\text{tg } 3x = \frac{3\text{tg } x - \text{tg}^3 x}{1 - 3\text{tg}^2 x}$, откуда

$\text{tg } x = \frac{3\text{tg}(x/3) - \text{tg}^3(x/3)}{1 - 3\text{tg}^2(x/3)}$. В качестве условия остановки рекурсии примем

условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,0001. Для вычисления функции $\text{tg}(x)$ 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. $\pi/4$ и $\pi/3$ (в радианах примерно 0,7854 и 1,0472), с точностью 0,0001. Вычисленные и контрольные значения совпадают.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте точность вычисления: 0.0001
Контрольное значение tan(0.7854) = 1.0000
Вычисленное значение myTan(0.7854) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.0472
Задайте точность вычисления: 0.0001
Контрольное значение tan(1.0472) = 1.7321
Вычисленное значение myTan(1.0472) = 1.7321
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

21. Предикат, вычисляющий функцию $\text{tg}(x)$, используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции $\text{tg}(x)$ с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу

$$\text{tg } x = \sum_{n=1}^{\infty} \frac{(-1)^{(n+1)} \cdot 2^{2n} \cdot (2^{2n} - 1) \cdot B_{2n}}{2n!} \cdot x^{2n-1}.$$

В качестве условия остановки рекурсии примем условие, что при $n=1$, согласно формуле разложения, $\text{tg } x = x$.

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 B_{2n} , 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. $\pi/4$ (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.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте количество членов ряда: 3
Контрольное значение tg(0.7854) = 1.0000
Вычисленное значение myTanR(0.7854) = 0.9867
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте количество членов ряда: 6
Контрольное значение tg(0.7854) = 1.0000
Вычисленное значение myTanR(0.7854) = 0.9998
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

22. 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 10 равно $3 \cdot 6 \cdot 9 = 162$.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите натуральное число: 10
Произведение чисел, кратных 3 и не превосходящих 10 = 162
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>

```

23. 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

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} \cdot x^n}{n!}, \text{ причем ряд сходится абсолютно на интервале } (-1, 1). \text{ В}$$

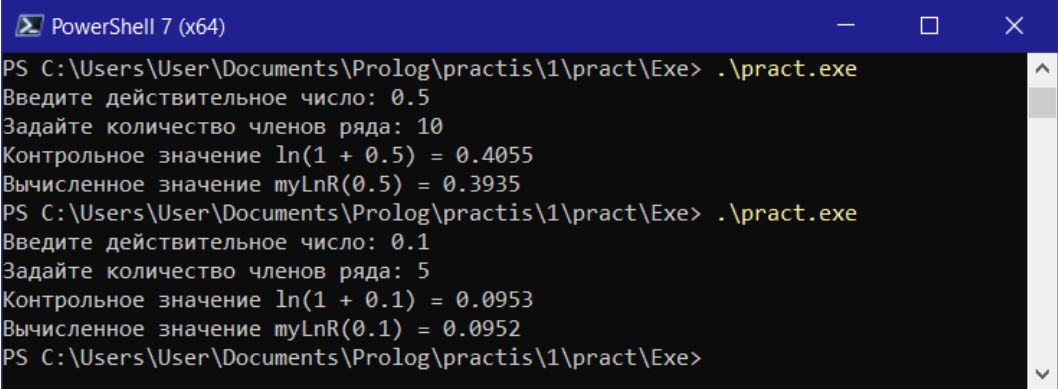
качестве условия остановки рекурсии примем условие, что при $n=1$, согласно формуле разложения, $\ln(1+x) = x$. Дополнительно, для вычисления используем предикат вычисления факториала числа `factor(integer, integer [out])`, принимающий аргумент как целое число и возвращающий результат вторым аргументом. 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.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.5
Задайте количество членов ряда: 10
Контрольное значение ln(1 + 0.5) = 0.4055
Вычисленное значение myLnR(0.5) = 0.3935
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.1
Задайте количество членов ряда: 5
Контрольное значение ln(1 + 0.1) = 0.0953
Вычисленное значение myLnR(0.1) = 0.0952
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>
```

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

Повтор 23-го задания.

25. Предикат, вычисляющий функцию $\text{ctg}(x)$ с заданной точностью.

Для вычисления функции $\text{ctg}(x)$ с заданной точностью используем формулу котангенс тройного угла $\text{ctg } 3x = \frac{3 \text{ctg}^2 x - 1}{\text{ctg}^3 x - 3 \text{ctg } x}$, откуда

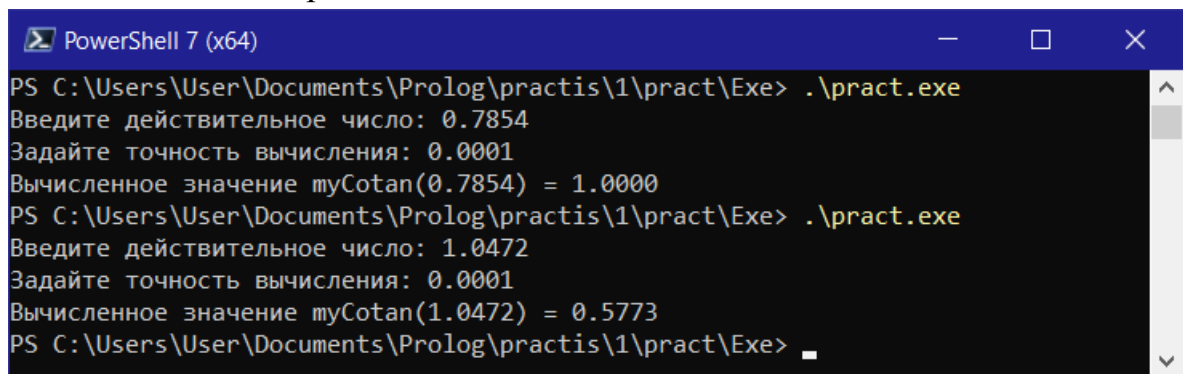
$$\text{ctg } x = \frac{3 \text{ctg}^2(x/3) - 1}{\text{ctg}^3(x/3) - 3 \text{ctg}(x/3)}.$$

В качестве условия остановки рекурсии примем условие, что вычисления заканчиваются, когда достигается требуемая точность, например, вычисленное значение меньше 0,0001. Для вычисления функции $\text{ctg}(x)$ 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. $\pi/4$ и $\pi/3$ (в радианах примерно 0,7854 и 1,0472), с точностью 0,0001. Вычисленные и контрольные значения совпадают.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте точность вычисления: 0.0001
Вычисленное значение myCotan(0.7854) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.0472
Задайте точность вычисления: 0.0001
Вычисленное значение myCotan(1.0472) = 0.5773
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> _
```

26. Предикат, вычисляющий функцию $\text{ctg}(x)$, используя разложение в ряд по заданному количеству членов ряда.

Для вычисления функции $\text{ctg}(x)$ с использованием разложения в ряд по заданному количеству членов ряда используем следующую формулу $\text{ctg } x = \sum_{n=1}^{\infty} \frac{(-1)^n \cdot 2^{2n} \cdot B_{2n}}{2n!} \cdot x^{2n-1}$. В качестве условия остановки рекурсии примем

условие, что при $n=0$, согласно формуле разложения, $\text{ctg } x = 1/x$. 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 B_{2n} , 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:

```
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 45 градусов, т. е. $\pi/4$ (в радианах примерно 0,7854), с вычислением 5 членов ряда. Полученное значение не отличается от контрольного.

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

```
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 0.7854
Задайте количество членов ряда: 5
Контрольное значение ctg(0.7854) = 1.0000
Вычисленное значение myCotanR(0.7854) = 1.0000
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe> .\pract.exe
Введите действительное число: 1.0472
Задайте количество членов ряда: 3
Контрольное значение ctg(1.0472) = 0.5773
Вычисленное значение myCotanR(1.0472) = 0.5777
PS C:\Users\User\Documents\Prolog\practis\1\pract\Exe>
```


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 списочные домены списка целых чисел `intList` и списка строк `srtList`:

```
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, [_ | T1], [_ | 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].

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
The list: [1,2,3,4,5]
Replaced list: [1,2,7,4,5]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

2. 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].

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
The list: [1,2,3,4,3]
Replaced all list: [1,2,7,4,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

3. 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].

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Введите натуральное число: 7
Список от 1 до 7: [1,2,3,4,5,6,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

4. 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].

```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Введите натуральное число: 9  
Список от 1 до 9: [9,8,7,6,5,4,3,2,1]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

5. 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.

```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Введите натуральное число: 6  
Список случайных чисел из 6 элементов: [79,14,21,61,19,22]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Введите натуральное число: 6  
Список случайных чисел из 6 элементов: [78,76,87,20,75,8]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

6. 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.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Введите количество элементов: 10
Введите начало промежутка: 20
Введите окончание промежутка: 60
Список случайных чисел из 10 элементов в промежутке от 20 до 60:
[50, 34, 23, 26, 48, 53, 45, 43, 47, 20]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Введите количество элементов: 10
Введите начало промежутка: 20
Введите окончание промежутка: 60
Список случайных чисел из 10 элементов в промежутке от 20 до 60:
[39, 55, 33, 32, 44, 40, 50, 54, 41, 46]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

7. 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.

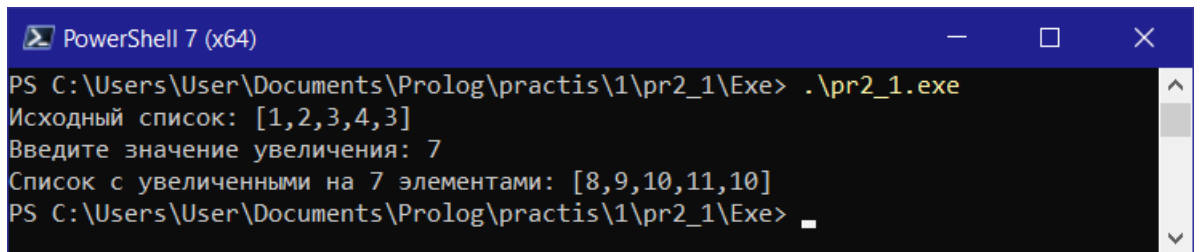
```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Введите начало промежутка: 10  
Введите окончание промежутка: 20  
Список случайных чисел из элементов в промежутке от 10 до 20:  
[12,15,17,13,18,17,10,10,15]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Введите начало промежутка: 10  
Введите окончание промежутка: 20  
Список случайных чисел из элементов в промежутке от 10 до 20:  
[13,18,14,10]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> _
```

8. 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].



```
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Исходный список: [1,2,3,4,3]
Введите значение увеличения: 7
Список с увеличенными на 7 элементами: [8,9,10,11,10]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

9. 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 = "X3"
    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 «X3», если число не представить как цифру: ["Шесть", "Три", "Ноль", "Девять", "X3"].

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список цифр: [6,3,0,9,45]
Список цифр словами: ["Шесть","Три","Ноль","Девять","ХЗ"]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

10. 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"].

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список цифр: [6,3,1,9,8]
Список римских цифр: ["VI","III","I","IX","VIII"]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

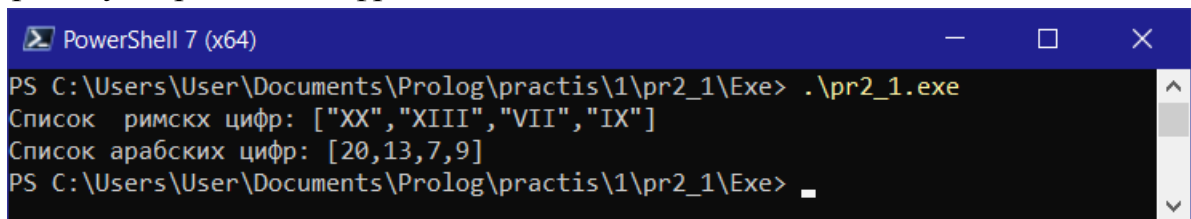
11. Предикат, переводящий список римских чисел в список соответствующих им арабских чисел (диапазон от 1 до 20).

Определим предикат `translateR2A : (strList, intList [out])`, принимающий список строк с римскими цифрами и выводящий список с арабскими цифрами. Определим вспомогательный предикат `rom2Arab : (string, integer`

[out]), принимающий строку с римской цифрой и выводящий соответствующую ей арабскую. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
rom2Arab(R, A) :-
    if R = "I" then A = 1
    elseif R = "II" then A = 2
    elseif R = "III" then A = 3
    elseif R = "IV" then A = 4
    elseif R = "V" then A = 5
    elseif R = "VI" then A = 6
    elseif R = "VII" then A = 7
    elseif R = "VIII" then A = 8
    elseif R = "IX" then A = 9
    elseif R = "X" then A = 10
    elseif R = "XI" then A = 11
    elseif R = "XII" then A = 12
    elseif R = "XIII" then A = 13
    elseif R = "XIV" then A = 14
    elseif R = "XV" then A = 15
    elseif R = "XVI" then A = 16
    elseif R = "XVII" then A = 17
    elseif R = "XVIII" then A = 18
    elseif R = "XIX" then A = 19
    elseif R = "XX" then A = 20
    else A = 0
end if.
translateR2A([], []) :- !.
translateR2A([DH | DT], [SH | Out]) :-
    rom2Arab(DH, SH), translateR2A(DT, Out).
run() :-
    RL = ["XX", "XIII", "VII", "IX"],
    stdio::writef("Список римских цифр: %", RL), nl,
    translateR2A(RL, L11),
    stdio::writef("Список арабских цифр: %", L11),
    _ = stdio::readLine().
```

При тестировании предиката задаем список римских цифр, таких как XX, XIII, VII и IX, которые переводятся как 20, 13, 7 и 9, что соответствует переводу в арабские цифры.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список римских цифр: ["XX","XIII","VII","IX"]
Список арабских цифр: [20,13,7,9]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> 
```

12. Предикат, удваивающий значения элементов списка.

Определим предикат `doubler` : (intList, intList [out]), принимающий список чисел и выводящий список с удвоенными числами. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
doubler([H | T], [H * 2 | NT]) :-
    doubler(T, NT).
doubler([], []).
```

```
run() :-
    L = [6, 3, 1, 9, 8],
    stdio::writef("Список чисел: %", L), nl,
    doubler(L, L11),
    stdio::writef("Список удвоенных чисел: %", L11),
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, 3, 1, 9 и 8. При удвоении должен получиться результат $6 \times 2 = 12$, $3 \times 2 = 6$, $1 \times 2 = 2$, $9 \times 2 = 18$ и $8 \times 2 = 16$, что соответствует выводу программы 12, 6, 2, 18, 16.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,3,1,9,8]
Список удвоенных чисел: [12,6,2,18,16]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

13. Предикат, преобразующий список, элементами которого являются числа, в список, элементы которого неотрицательны.

Определим предикат `positiv : (intList, intList [out])`, принимающий список чисел и выводящий список с неотрицательными числами. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
positiv([H | T], [math::abs(H) | NT]) :-
    positiv(T, NT).
positiv([], []).

run() :-
    L = [6, -3, 1, 9, -8],
    stdio::writef("Список чисел: %", L),
    nl,
    positiv(L, L11),
    stdio::writef("Список неотрицательных чисел: %", L11),
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9 и -8, которые преобразуются в неотрицательные 6, 3, 1, 9 и 8, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8]
Список неотрицательных чисел: [6,3,1,9,8]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

14. Предикат, преобразующий исходный список в список позиций отрицательных элементов.

Определим предикат `negPos : (intList, intList [out]) determ`, принимающий список чисел и выводящий список с позициями отрицательных элементов. Дополнительно определим предикат `negPosImp : (intList, integer, intList [out]) determ`, выполняющий ту же задачу, но использующий аргумент для хранения значения счетчика позиций. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```

negPosImp([], _, []) :- !.
negPosImp([H | T], K, [K | T1]) :- H < 0, !, K1 = K + 1,
    negPosImp(T, K1, T1).
negPosImp([H | T], K, T1) :- H >= 0, !, K1 = K + 1, negPosImp(T, K1, T1).
negPos(L, R) :- negPosImp(L, 1, R).

run() :-
    L = [6, -3, 1, 9, -8],
    stdio::writef("Список чисел: %", L), nl,
    negPos(L, L14),
    stdio::writef("Список позиций отрицательных чисел: %", L14),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9 и -8. Имеются отрицательные элементы -3 и -8, занимающие позиции 2 и 5, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8]
Список позиций отрицательных чисел: [2,5]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

15. Предикат, удаляющий из исходного списка элементы с четными номерами.

Определим предикат `delEven : (intList, intList [out])`, принимающий список чисел и выводящий исходный список с удаленными элементами, занимавших позиции с четными номерами. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```

delEven([], []).
delEven([X], [X]) :- !.
delEven([X, _ | T], [X | T2]) :- delEven(T, T2).

run() :-
    L = [6, -3, 1, 9, -8],
    stdio::writef("Список чисел: %", L), nl,
    delEven(L, L15),
    stdio::writef("Список без четных элементов: %", L15),
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9 и -8. На позициях с четными номерами находятся числа -3 и 9. Без них исходный список имеет вид 6, 1 и -8, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8]
Список без четных элементов: [6,1,-8]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

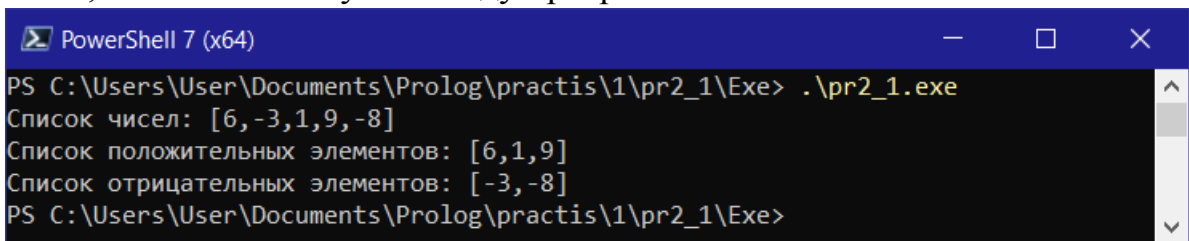
16. Предикат, который разделит исходный список из целых чисел на два списка: список положительных чисел и список отрицательных чисел.

Определим предикат `splitPosNeg : (intList, intList [out], intList [out])`, принимающий список чисел и выполняющий разделение исходного списка на два: 2-ой выходной аргумент — список отрицательных элементов, 3-ий выходной аргумент — список неотрицательных элементов. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
splitPosNeg([], [], []).
splitPosNeg([H | T], [H | T1], T2) :- H < 0, !, splitPosNeg(T, T1, T2).
splitPosNeg([H | T], T1, [H | T2]) :- splitPosNeg(T, T1, T2).

run() :-
    L = [6, -3, 1, 9, -8],
    stdio::writef("Список чисел: %", L), nl,
    splitPosNeg(L, L16N, L16P),
    stdio::writef("Список положительных элементов: %\n", L16P),
    stdio::writef("Список отрицательных элементов: %", L16N),
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9 и -8. Список отрицательных элементов состоит из -3 и -8, а положительных из 6, 1 и 9, что соответствует выводу программы.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8]
Список положительных элементов: [6,1,9]
Список отрицательных элементов: [-3,-8]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

17. Предикат, разделяющий исходный список на два подсписка. В первый из них должны попасть элементы с нечетными номерами, во второй — элементы с четными номерами.

Определим предикат `splitOddEven : (intList, intList [out], intList [out]) nondeterm`, принимающий список чисел и выполняющий разделение исходного списка на два: 2-ой выходной аргумент — список элементов на нечетных позициях, 3-ий выходной аргумент — список элементов на четных позициях. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
splitOddEven([X], [X], []).
splitOddEven([X, Y], [X], [Y]).
splitOddEven([X, Y | T], [X | T1], [Y | T2]) :- splitOddEven(T, T1, T2).

run() :-
    L = [6, -3, 1, 9, -8],
    stdio::writef("Список чисел: %", L), nl,
    splitOddEven(L, L17O, L17E),
    stdio::writef("Список элементов с Нечетными номерами позиции: %\n",
        L17O),
    stdio::writef("Список элементов с четными номерами позиции: %", L17E),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9 и -8. На нечетных позициях находятся элементы 6, 1 и 9, на четных — -3 и 9, что соответствует выводу программы.

```
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8]
Список элементов с НЕчетными номерами позиции: [6,1,-8]
Список элементов с четными номерами позиции: [-3,9]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

18. Предикат, вычисляющий по списку и числу подсписок исходного списка, начинающийся с элемента с указанным номером.

Определим предикат `sublist : (intList, integer, intList [out]) nondeterm`, принимающий список чисел, номер позиции, и выводящий подсписок, начинающийся с указанной позиции исходного списка. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
sublist(X, 0, X).
sublist([_ | T], N, R) :- N1 = N - 1, sublist(T, N1, R).

run() :-
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],
    stdio::writef("Список чисел: %", L), nl,
    stdio::write("Введите номер элемента (нумерация начинается с 0): "),
    N = stdio::read(), hasDomain(integer, N),
    sublist(L, N, L18),
    stdio::writef("Подсписок начиная позиции № %: %", N, L18),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13 и вводим номер позиции: 7. Так как условием остановки рекурсии задана позиция равная 0, нумерация элементов начинается с 0. Поэтому на 7-ой позиции в исходном списке находится число 7 и подсписок состоит из чисел 7 и -13, что соответствует выводу программы.

При очередном тестировании предиката вводим номер позиции: 3. На 3-ей позиции в исходном списке находится число 9 и подсписок состоит из чисел 9, -8, 14, 23, 7 и -13, что соответствует выводу программы.

```
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите номер элемента (нумерация начинается с 0): 7
Подсписок начиная позиции № 7: [7,-13]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите номер элемента (нумерация начинается с 0): 3
Подсписок начиная позиции № 3: [9,-8,14,23,7,-13]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```


19. Предикат, осуществляющий удаление указанного количества последних элементов исходного списка.

Определим предикат `delLast : (integer, intList, intList [out]) determ`, принимающий количество элементов для удаления, исходный список чисел, и выводящий исходный список без указанного количества последних элементов. Для работы предиката определены дополнительные предикаты: `del : (intList, intList [out]) determ` — удаляет первый элемент списка; `delFirst : (integer, intList, intList [out]) determ` — удаляет указанное количество первых элементов списка; `reverseImp : (intList, intList, intList [out])` — реализация реверса списка с дополнительным аргументом; `reverse : (intList, intList [out])` — реверс списка чисел. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
del([_ | T], T).
delFirst(1, S, R) :- del(S, R), !.
delFirst(N, S, R) :- N1 = N - 1, del(S, R1), delFirst(N1, R1, R).
reverseImp([H | L1], L2, R) :- reverseImp(L1, [H | L2], R).
reverseImp([], L, L).
reverse(L, R) :- reverseImp(L, [], R).
delLast(N, S, R) :- reverse(S, R1), delFirst(N, R1, R2), reverse(R2, R).

run() :-
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],
    stdio::writef("Список чисел: %", L), nl,
    stdio::write("Введите количество удаляемых с конца элементов: "),
    N = stdio::read(), hasDomain(integer, N),
    delLast(N, L, L19),
    stdio::writef("Результат удаления % элементов:\n%", N, L19),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13 и вводим количество удаляемых элементов: 3. Три последних элемента — это 23, 7 и -13: без них исходный список имеет вид 6, -3, 1, 9, -8, 14, что соответствует выводу программы.

При очередном тестировании предиката вводим количество удаляемых элементов: 5. Пять последних элементов — это -8, 14, 23, 7 и -13: без них исходный список имеет вид 6, -3, 1, 9, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите количество удаляемых с конца элементов: 3
Результат удаления 3 элементов:
[6,-3,1,9,-8,14]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите количество удаляемых с конца элементов: 5
Результат удаления 5 элементов:
[6,-3,1,9]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> _
```

20. Предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов из начала списка, во второй – оставшиеся элементы.

Определим предикат `splitByPos` : (intList, integer, intList [out], intList [out]) determ, принимающий список чисел, количество элементов начала списка, и выполняющий разделение исходного списка на два: 3-ий выходной аргумент — список указанного количества элементов начала списка, 4-ый выходной аргумент — оставшиеся элементы списка. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
splitByPos(L, 0, [], L) :- !.  
splitByPos([H | T], N, [H | Before], After) :-  
    N1 = N - 1, splitByPos(T, N1, Before, After).  
  
run() :-  
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],  
    stdio::writef("Список чисел: %", L), nl,  
    stdio::write("Введите номер позиции для разделения: "),  
    N = stdio::read(), hasDomain(integer, N),  
    splitByPos(L, N, L20B, L20A),  
    stdio::writef("Список до % позиции (включительно): %\n", N, L20B),  
    stdio::writef("Список после % позиции: %", N, L20A),  
    fail;  
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13 и вводим количество элементов первого списка: 4. Четыре первых элемента списка — это 6, -3, 1 и 9; остальной список — -8, 14, 23, 7 и -13, что соответствует выводу программы.

При очередном тестировании предиката вводим количество элементов первого списка: 7. Семь первых элементов списка — это 6, -3, 1, 9, -8, 14 и 23; остальной список — 7 и -13, что соответствует выводу программы.

```
PowerShell 7 (x64)  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Список чисел: [6,-3,1,9,-8,14,23,7,-13]  
Введите номер позиции для разделения: 4  
Список до 4 позиции (включительно): [6,-3,1,9]  
Список после 4 позиции: [-8,14,23,7,-13]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe  
Список чисел: [6,-3,1,9,-8,14,23,7,-13]  
Введите номер позиции для разделения: 7  
Список до 7 позиции (включительно): [6,-3,1,9,-8,14,23]  
Список после 7 позиции: [7,-13]  
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> _
```

21. Предикат, осуществляющий разделение исходного списка на два подсписка. В первый из них должно попасть указанное количество элементов с конца списка, во второй – оставшиеся элементы.

Определим предикат `splitByEndPos` : (intList, integer, intList [out], intList [out]) determ, принимающий список чисел, количество элементов с конца списка, и выполняющий разделение исходного списка на два: 3-ий выходной

аргумент — список указанного количества элементов с конца списка, 4-ый выходной аргумент — оставшиеся начальные элементы списка. Для реализации предиката используются ранее определенные предикаты `splitByPos` и `reverse` из 20-го и 19-го заданий соответственно. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
splitByEndPos(L, N, R1, R2) :- reverse(L, T), splitByPos(T, N, T1, T2),
    reverse(T1, R1), reverse(T2, R2).

run() :-
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],
    stdio::writef("Список чисел: %", L), nl,
    stdio::write("Введите номер позиции (с конца) для разделения: "),
    N = stdio::read(), hasDomain(integer, N),
    splitByEndPos(L, N, L21B, L21A),
    stdio::writef("Список после % позиции (включительно): %\n", N, L21B),
    stdio::writef("Список до % позиции: %", N, L21A),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13 и вводим необходимое количество элементов с конца списка: 3. Три последних элемента списка — это 23, 7 и -13; оставшийся начальный список — 6, -3, 1, 9, -8 и 14, что соответствует выводу программы.

При очередном тестировании предиката вводим необходимое количество элементов с конца списка: 7. Семь последних элементов списка — это 1, 9, -8, 14, 23, 7 и -13; оставшийся начальный список — 6 и -3, что соответствует выводу программы.

```
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите номер позиции (с конца) для разделения: 3
Список после 3 позиции (включительно): [23,7,-13]
Список до 3 позиции: [6,-3,1,9,-8,14]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Введите номер позиции (с конца) для разделения: 7
Список после 7 позиции (включительно): [1,9,-8,14,23,7,-13]
Список до 7 позиции: [6,-3]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> _
```

22. Предикат, находящий предпоследний элемент списка.

Определим предикат `getPreLast : (intList, integer [out]) determ`, принимающий исходный список чисел, и выводящий предпоследний элемент исходного списка. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
getPreLast([H, _], H) :- !.
getPreLast([_ | T], R) :- getPreLast(T, R).

run() :-
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],
    stdio::writef("Список чисел: %", L), nl,
```

```

getPreLast(L, R),
stdio::writef("Предпоследний элемент в списке равен: %", R),
fail;
_ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13. Предпоследним элементом списка является число 7, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Предпоследний элемент в списке равен: 7
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

23. Предикат, удаляющий предпоследний элемент списка.

Определим предикат `delPreLast : (intList, intList [out]) determ`, принимающий исходный список чисел, и выводящий исходный список без предпоследнего элемента. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```

delPreLast([], _) :- fail.
delPreLast([_, X], [X]) :- !.
delPreLast([H | T], [H | R]) :- delPreLast(T, R).

run() :-
    L = [6, -3, 1, 9, -8, 14, 23, 7, -13],
    stdio::writef("Список чисел: %", L), nl,
    delPreLast(L, L23),
    stdio::writef("Список без предпоследнего элемента: %", L23),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, -8, 14, 23, 7, -13. Предпоследним элементом списка является число 7, поэтому исходный список без предпоследнего элемента будет иметь вид 6, -3, 1, 9, -8, 14, 23, -13, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,-8,14,23,7,-13]
Список без предпоследнего элемента: [6,-3,1,9,-8,14,23,-13]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

24. Предикат, заменяющий в исходном списке два подряд идущих одинаковых элемента одним.

Определим предикат `delDouble : (intList, intList [out])`, принимающий список чисел, и выводящий исходный список с замененными два раза подряд идущими одинаковыми элементами одним. Данный предикат реализован через дополнительный предикат реализации `delDoubleImp : (integer, intList, intList [out])`, принимающий в качестве первого аргумента счетчик повторений. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```

delDoubleImp(A, [A | B], C) :- !, delDoubleImp(A, B, C).
delDoubleImp(_, [A | B], [A | C]) :- !, delDoubleImp(A, B, C).
delDoubleImp(_, _, []).
delDouble(L, R) :- delDoubleImp(1, L, R).

run() :-
    L = [6, -3, 1, 9, 9, 14, 23, 7, 7],
    stdio::writef("Список чисел: %", L), nl,
    delDouble(L, L24),
    stdio::writef("Список без дубликатов: %", L24),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 9, 9, -8, 14, 23, 7, 7. В списке присутствуют два раза подряд идущие одинаковые элементы, такие как 9 на 4-ой и 5-ой позициях (нумерация с 1) и 7 на 9-ой и 10-ой позициях. При устранении дубликатов получим список 6, -3, 1, 9, -8, 14, 23, 7, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,9,9,14,23,7,7]
Список без дубликатов: [6,-3,1,9,14,23,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

25. Предикат, удаляющий в исходном списке все повторные вхождения элементов.

Определим предикат `setof` : (intList, intList [out]) nondeterm, принимающий список чисел, и выводящий исходный список с удаленными повторными вхождениями элементов. Данный предикат реализован через дополнительный предикат `mem` : (integer, intList) determ, проверяющий принадлежность числа списку чисел. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```

mem(_, []) :- fail.
mem(X, [X | _]) :- !.
mem(X, [_ | T]) :- mem(X, T).
setof([], []).
setof([H | T], [H | R]) :- setof(T, R), not(mem(H, R)).
setof([H | T], R) :- setof(T, R), mem(H, R).

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список чисел: %", L), nl,
    setof(L, L25),
    stdio::writef("Список без дубликатов: %", L25),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7. В списке присутствуют три элемента со значением 9 и два элемента со значением 7. При удалении повторных вхождений элементов, оставляя последнее, получим список 6, -3, 1, 14, 9, 23, 7, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,7,9,9,14,9,23,7,7]
Список без дубликатов: [6,-3,1,14,9,23,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

26. Предикат, осуществляющий перестановку двух элементов списка с заданными номерами.

Определим предикат `remove : (integer, integer, intList, intList [out]) determ`, принимающий два значения номеров позиций чисел, исходный список чисел, и выводящий исходный список с переставленными значениями. Данный предикат реализован через дополнительный предикат `element : (integer, integer, integer [out], intList, intList [out]) determ`, находящий позицию элемента в списке чисел. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
element(1, A, B, [B | L], [A | L]) :- !.
element(Y, A, B, [C | L], [C | L1]) :- Y1 = Y - 1,
    element(Y1, A, B, L, L1).
remove(X, X, L, L) :- !.
remove(X, Y, [A | L], [A | L1]) :- X > 1, X < Y, !,
    X1 = X - 1, Y1 = Y - 1, remove(X1, Y1, L, L1).
remove(1, Y, [A | L], [B | L1]) :- !, Y1 = Y - 1,
    element(Y1, A, B, L, L1).
remove(X, Y, L, R) :- Y > 0, remove(Y, X, L, R).

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список чисел: %", L), nl,
    stdio::write("Введите номер 1-го элемента: "),
    N = stdio::read(), hasDomain(integer, N),
    stdio::write("Введите номер 2-го элемента: "),
    M = stdio::read(), hasDomain(integer, M),
    remove(N, M, L, L26),
    stdio::writef("Список перестановки элементов: %", L26),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7 и вводим номера элементов для перестановки: 1 и 2 (нумерация с 1). Первый и второй элементы — это 6 и -3; меняя их местами получим список — -3, 6, 1, 7, 9, 9, 14, 9, 23, 7, 7, что соответствует выводу программы.

При очередном тестировании предиката вводим номера элементов для перестановки: 1 и 11. Первый и одиннадцатый элементы — это 6 и 7; меняя их местами получим список — 7, -3, 1, 7, 9, 9, 14, 9, 23, 7, 6, что соответствует выводу программы.


```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,7,9,9,14,9,23,7,7]
Введите номер 1-го элемента: 1
Введите номер 2-го элемента: 2
Список перестановки элементов: [-3,6,1,7,9,9,14,9,23,7,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список чисел: [6,-3,1,7,9,9,14,9,23,7,7]
Введите номер 1-го элемента: 1
Введите номер 2-го элемента: 11
Список перестановки элементов: [7,-3,1,7,9,9,14,9,23,7,6]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

27. Предикат, генерирующий все перестановки элементов списка, указанного в качестве первого аргумента предиката.

Определим предикат `permAll : (strList) failure`, принимающий список строк и генерирующий все перестановки элементов исходного списка. Для работы предиката определены дополнительные предикаты: `place : (string, strList, strList [out]) multi` — проверяет, является ли 3-ий аргумент списком, полученным вставкой 1-го значения в произвольное место 2-го аргумента; `perm : (strList, strList [out]) multi` — генерирует списки, полученные перестановкой элементов 1-го аргумента; `permAll : (strList) failure` — выводит все возможные перестановки элементов списка. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
place(X, L, [X | L]).
place(X, [H | L], [H | R]) :- place(X, L, R).
perm([], []).
perm([H | L], Z) :- perm(L, Y), place(H, Y, Z).
permAll(L) :- perm(L, R), stdio::write(R), nl, fail.

run() :-
    L = ["привет", "Жорик", "как дела"],
    stdio::writef("Список: %", L), nl,
    stdio::write("Все перестановки:\n"),
    permAll(L), fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список строк, таких как "привет", "Жорик" и "как дела". Выполняя перестановки элементов списка, получим: "привет", "Жорик", "как дела"; "Жорик", "привет", "как дела"; "Жорик", "как дела", "привет"; "привет", "как дела", "Жорик"; "как дела", "привет", "Жорик"; "как дела", "Жорик", "привет"; , что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: ["привет","Жорик","как дела"]
Все перестановки:
["привет","Жорик","как дела"]
["Жорик","привет","как дела"]
["Жорик","как дела","привет"]
["привет","как дела","Жорик"]
["как дела","привет","Жорик"]
["как дела","Жорик","привет"]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

28. Предикат, осуществляющий циклический сдвиг элементов списка на один влево (вправо).

Определим предикат `shift : (integer, intList, intList [out]) nondeterm`, принимающий число как направление сдвига (> 0 — вправо, < 0 — влево), исходный список чисел, и выводящий список со сдвигом элементов. Для работы предиката определен дополнительный предикат: `conc : (intList, intList, intList [out])` — добавляет в конец 1-го списка элементы 2-го списка и выводит получившийся список в 3-ий аргумент. Также используются ранее определенные предикаты `splitByPos` и `splitByEndPos` из 20 и 21 заданий соответственно. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
conc([], L, L).
conc([H | T], L, [H | L1]) :- conc(T, L, L1).
shift(Direction, L, R) :- Direction > 0,
    splitByEndPos(L, 1, End, Begin), conc(End, Begin, R), !;
    splitByPos(L, 1, Begin, End), conc(End, Begin, R), !.

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    stdio::write("Введите направление сдвига (1 - вправо, -1 - влево): "),
    N = stdio::read(), hasDomain(integer, N),
    shift(N, L, L28),
    stdio::writef("Список после сдвига: %", L28),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7 и вводим направление сдвига вправо: 1. При сдвиге вправо 1-ый элемент займет позицию 2-го, 2-ой — позицию 3-го и т. д., а последний — позицию 1-го. Получим список 7, 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, что соответствует выводу программы.

При очередном тестировании предиката вводим направление сдвига влево: -1. При сдвиге влево 2-ой элемент займет позицию 1-го, 3-ий — позицию 2-го и т. д., а 1-ый — позицию последнего. Получим список -3, 1, 7, 9, 9, 14, 9, 23, 7, 7, 6, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Введите направление сдвига (1 - вправо, -1 - влево): 1
Список после сдвига: [7,6,-3,1,7,9,9,14,9,23,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Введите направление сдвига (1 - вправо, -1 - влево): -1
Список после сдвига: [-3,1,7,9,9,14,9,23,7,7,6]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

29. Предикат, осуществляющий циклический сдвиг элементов списка на заданное количество шагов влево (вправо).

Определим предикат `shiftN` : (integer Direction, integer Shift, intList Source, intList Result [out]) nondeterm, принимающий 1-ое число как направление сдвига (> 0 — вправо, < 0 — влево), 2-ое число как расстояние сдвига (на какое количество элементов сдвинуть), исходный список чисел, и выводящий список со сдвигом элементов. Для работы предиката используются ранее определенные предикаты `conc`, `splitByPos` и `splitByEndPos` из 28, 20 и 21 заданий соответственно. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
shiftN(Direction, Shift, Source, Result) :-
    Direction > 0, splitByEndPos(Source, Shift, End, Begin),
    conc(End, Begin, Result), !;
    splitByPos(Source, Shift, Begin, End), conc(End, Begin, Result), !.

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    stdio::write("Введите направление сдвига (1 - вправо, -1 - влево): "),
    D = stdio::read(), hasDomain(integer, D),
    stdio::write("Введите значение сдвига: "),
    S = stdio::read(), hasDomain(integer, S),
    shiftN(D, S, L, L29),
    stdio::writef("Список после сдвига: %", L29),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7 и вводим направление сдвига влево: -1 и значение сдвига: 2. При сдвиге влево на 2 элемента получим список 1, 7, 9, 9, 14, 9, 23, 7, 7, 6, -3, что соответствует выводу программы.

При очередном тестировании предиката вводим направление сдвига вправо: 1 и значение сдвига: 5. При сдвиге вправо на 5 элементов получим список 14, 9, 23, 7, 7, 6, -3, 1, 7, 9, 9, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Введите направление сдвига (1 - вправо, -1 - влево): -1
Введите значение сдвига: 2
Список после сдвига: [1,7,9,9,14,9,23,7,7,6,-3]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Введите направление сдвига (1 - вправо, -1 - влево): 1
Введите значение сдвига: 5
Список после сдвига: [14,9,23,7,7,6,-3,1,7,9,9]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

30. Предикат, осуществляющий поэлементное перемножение соответствующих элементов двух исходных списков.

Определим предикат `multy : (intList A, intList B, intList Result [out]) determ`, принимающий два списка чисел, и выводящий список с поэлементно перемноженными соответствующими элементами двух исходных списков. Данный предикат объявляется в секции `class predicates`, а определяется в секции `clauses`, код которой приведен ниже:

```
multy([], [], []).
multy([H1 | T1], [H2 | T2], [H1 * H2 | Result]) :- multy(T1, T2, Result).

run() :-
    L1 = [1, 2, 3],
    stdio::writef("Первый список: %", L1), n\,
    L2 = [4, 5, 3],
    stdio::writef("Второй список: %", L2), n\,
    multy(L1, L2, L30),
    stdio::writef("Результат перемножения поэлементно: %", L30),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем 2 списка чисел, такие как 1, 2, 3 и 4, 5, 3. При поэлементном перемножении списков получим $1 \times 4 = 4$, $2 \times 5 = 10$, $3 \times 3 = 9$, то есть список 4, 10, 9, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Первый список: [1,2,3]
Второй список: [4,5,3]
Результат перемножения поэлементно: [4,10,9]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

31. Предикат, вычисляющий скалярное произведение векторов, заданных списками целых чисел.

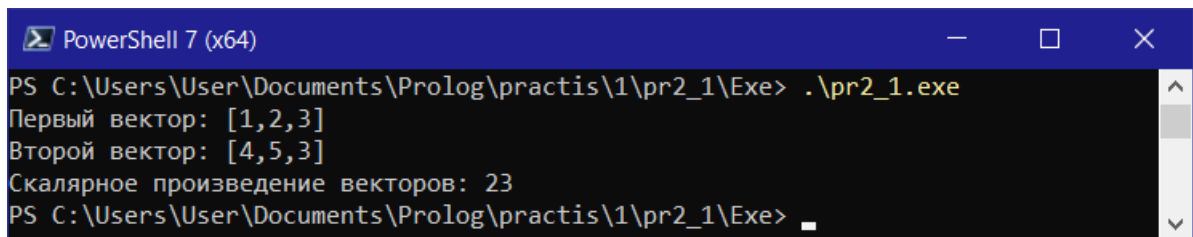
Определим предикат `scalarPro : (intList A, intList B, integer Result [out]) determ`, принимающий два вектора чисел в виде списков, и выводящий результат скалярного произведения этих векторов. Для работы предиката определен дополнительный предикат: `sumIntList : (intList L, integer Result [out])` — вычисляет значение суммы элементов списка чисел и выводит это значение в 3-ий аргумент. Также используется ранее определенный предикат

multy из 30 задания. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

```
sumIntList([], 0) :- !.
sumIntList([H | T], R) :- sumIntList(T, R1), R = R1 + H.
scalarPro(A, B, R) :- multy(A, B, R1), sumIntList(R1, R).

run() :-
    L1 = [1, 2, 3],
    stdio::writef("Первый вектор: %", L1), nl,
    L2 = [4, 5, 3],
    stdio::writef("Второй вектор: %", L2), nl,
    scalarPro(L1, L2, L31),
    stdio::writef("Скалярное произведение векторов: %", L31),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем 2 вектора чисел, такие как 1, 2, 3 и 4, 5, 3. При поэлементном перемножении векторов получим $1 \times 4 = 4$, $2 \times 5 = 10$, $3 \times 3 = 9$, то есть список значений 4, 10, 9. Сложив все элементы этого списка получим значение скалярного произведения равное 23, что соответствует выводу программы.



```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Первый вектор: [1,2,3]
Второй вектор: [4,5,3]
Скалярное произведение векторов: 23
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> █
```

32. 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.

Для работы с парами чисел и списками пар объявим соответствующие домены:

```
domains
    pair = p(integer, integer).
    pList = pair*.
```

Определим предикат culc : (intList, pList [out]), принимающий список чисел, и выводящий список пар со значениями элементов и количеством их вхождений в исходный список. Для работы предиката определен дополнительный предикат: into : (integer, pList, pList [out]) — добавляющий значение в список пар и увеличивающий счетчик количества вхождений. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

```
into(X, [], [p(X, 1)]) :- !.
into(X, [p(Y, N) | T], [p(Y, N1) | T]) :- X = Y, !, N1 = N + 1.
into(X, [H | T], [H | T1]) :- into(X, T, T1).
culc([], []) :- !.
culc([X], [p(X, 1)]) :- !.
culc([H | T], L) :- culc(T, T1), into(H, T1, L).
```

```

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    culc(L, L32),
    stdio::writef("Список элементов и числа их вхождения:\n%", L32),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7. При просмотре списка видно, что значение 7 повторяется 3 раза, 9 — также 3 раза, остальные элементы по 1 разу, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Список элементов и числа их вхождения:
[p(7,3),p(23,1),p(9,3),p(14,1),p(1,1),p(-3,1),p(6,1)]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

33. Предикат, определяющий первую позицию подсписка в списке.

Определим предикат `firstPosSublist` : (intList Source, intList SubList, integer Position [out]) determ, принимающий список чисел, искомый подсписок, и выводящий номер позиции подсписка в исходном списке. Для работы предиката определен дополнительный предикат: `length` : (intList, integer [out]) — вычисляет количество элементов списка чисел и выводит это значение во 2-ой аргумент. Также используется ранее определенный предикат `splitByPos` из 20 задания. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```

length([], 0).
length([_ | T], L) :- length(T, LT), L = LT + 1.

firstPosSublist(L, SL, 1) :-
    length(SL, Res), splitByPos(L, Res, R, _), R = SL, !.
firstPosSublist([_ | T], SL, R) :-
    firstPosSublist(T, SL, R1), R = R1 + 1.

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    SL = [14, 9, 23],
    stdio::writef("Подсписок: %", SL), nl,
    firstPosSublist(L, SL, R),
    stdio::writef("Первая позиция подсписка в списке: %", R),
    fail;
    _ = stdio::readLine().

```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7 и искомый подсписок 14, 9, 23. При просмотре списка видно, что значение 14 находится на 7-ой позиции (нумерация с 1). Далее за ним следуют значения 9 и 23, следовательно, искомый подсписок имеет начинается с 7 позиции, что соответствует выводу программы.


```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Подсписок: [14,9,23]
Первая позиция подсписка в списке: 7
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> _
```

34. Предикат, добавляющий элементы одного списка во второй список, начиная с заданной позиции.

Определим предикат `insertSublist : (intList Source, integer Position, intList Sublist, intList Result [out]) determ`, принимающий список чисел, позицию для вставки подсписка, сам подсписок, и выводящий исходный список со вставленным в указанной позиции подсписком. Для работы предиката используется ранее определенный предикат `conc` из 28 задания. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```
insertSublist(L, 1, SL, Result) :- conc(SL, L, Result), !.
insertSublist([H | T], Pos, SL, [H | Result]) :- NextPos = Pos - 1,
    insertSublist(T, NextPos, SL, Result).

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    SL = [14, 9, 23],
    stdio::writef("Подсписок: %", SL), nl,
    stdio::write("Введите номер позиции для вставки: "),
    N = stdio::read(), hasDomain(integer, N),
    insertSublist(L, N, SL, R),
    stdio::writef("Подсписок вставлен в список в % позиции: %", N, R),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7, подсписок для вставки 14, 9, 23 и номер позиции для вставки подсписка: 3. Добавив подсписок в исходный список так, чтобы 1-ый элемент подсписка занимал 3-ью позицию исходного списка, получим список 6, -3, 14, 9, 23, 1, 7, 9, 9, 14, 9, 23, 7, 7, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6,-3,1,7,9,9,14,9,23,7,7]
Подсписок: [14,9,23]
Введите номер позиции для вставки: 3
Подсписок вставлен в список в 3 позиции: [6,-3,14,9,23,1,7,9,9,14,9,23,7,7]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

35. Предикат, возвращающий по списку и двум числам M и N подсписок исходного списка, состоящий из элементов с номерами от M до N.

Определим предикат `sublistFromTo : (integer From, integer To, intList Source, intList Result [out]) nondeterm`, принимающий начальную и конечную позиции подсписка, список чисел, и выводящий подсписок из исходного списка от начальной до конечной позиции. Для работы предиката используются ранее определенные предикаты `reverse`, `conc`, `splitByPos` и

splitByEndPos из 19, 28, 20 и 21 заданий соответственно. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже::

```
sublistFromTo(M, N, L, R) :- splitByPos(L, N, Left, _),
    splitByEndPos(Left, N - M + 1, R, _).

run() :-
    L = [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7],
    stdio::writef("Список: %", L), nl,
    stdio::write("Введите номер позиции начала подсписка: "),
    M = stdio::read(), hasDomain(integer, M),
    stdio::write("Введите номер позиции окончания подсписка: "),
    N = stdio::read(), hasDomain(integer, N),
    sublistFromTo(M, N, L, R),
    stdio::writef("Подсписок от % до % позиции: %", M, N, R),
    fail;
    _ = stdio::readLine().
```

При тестировании предиката задаем список чисел, таких как 6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7 и вводим начало подсписка: 1 и окончание: 3. В исходном списке на позициях с 1-ой по 3-ью находятся элементы 6, -3, и 1, что соответствует выводу программы.

При очередном тестировании предиката вводим начало подсписка: 3 и окончание: 7. В исходном списке на позициях с 3-ей по 7-ую находятся элементы 1, 7, 9, 9, 14, что соответствует выводу программы.

```
PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7]
Введите номер позиции начала подсписка: 1
Введите номер позиции окончания подсписка: 3
Подсписок от 1 до 3 позиции: [6, -3, 1]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Список: [6, -3, 1, 7, 9, 9, 14, 9, 23, 7, 7]
Введите номер позиции начала подсписка: 3
Введите номер позиции окончания подсписка: 7
Подсписок от 3 до 7 позиции: [1, 7, 9, 9, 14]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>
```

36. Предикат, формирующий список простых чисел, не превосходящих данного числа.

Определим предикат primeList : (integer, intList [out]) multi, принимающий число, и выводящий список простых чисел, не превышающих введенное число. Для работы предиката определены дополнительные предикаты: factor : (integer, integer) determ — находит все делители числа, заданного 1-ым аргументом; isPrime : (integer) determ — определяет, является ли введенное число простым. Также используется ранее определенный предикат reverse из 19 задания. Данные предикаты объявляются в секции class predicates, а определяются в секции clauses, код которой приведен ниже:

```
factor(X, Y) :- X mod Y = 0, !.
factor(X, Y) :- Z = X * 1.000000000001, hasDomain(math::uReal, Z),
    math::sqrt(Z) > Y, Y1 = Y + 2, factor(X, Y1).
```

```

isPrime(N) :- N > 3, N mod 2 = 1, not(factor(N, 3)), !.
isPrime(3). isPrime(2). isPrime(1).

primeList(N, L) :- N <= 0, L = [];
    isPrime(N), N - 1 = NN, primeList(NN, LL), L = [N | LL];
    NN = N - 1, primeList(NN, L).

run() :-
    stdio::write("Введите натуральное число: "),
    M = stdio::read(), hasDomain(integer, M),
    primeList(M, R1), reverse(R1, R),
    stdio::writef("Список простых чисел, не превосходящих %:\n%", M, R), !,
    _ = stdio::readLine().

```

При тестировании предиката вводим число 49. Простыми числами, не превышающими 49, являются числа 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 и 47, что соответствует выводу программы.

При очередном тестировании предиката вводим число 101. Простыми числами, не превышающими 101, являются числа 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 и 101, что соответствует выводу программы.

```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Введите натуральное число: 49
Список простых чисел, не превосходящих 49:
[1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Введите натуральное число: 101
Список простых чисел, не превосходящих 101:
[1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```

37. Предикат, транспонирующий матрицу, заданную списком списков.

Определим предикат `transpose : (intList*, intList* [out]) nondeterm`, принимающий список списков чисел, являющийся матрицей чисел, и выводящий так же список списков чисел, являющийся транспонированной матрицей по отношению к 1-му аргументу. Для работы предиката определены дополнительные предикаты: `transpose_1st_col : (intList*, intList [out], intList* [out])` — транспонирует 1-ый столбец матрицы; `printMatrix : (intList*) determ` — выводит матрицу на печать построчно. Данные предикаты объявляются в секции `class predicates`, а определяются в секции `clauses`, код которой приведен ниже:

```

transpose([], [], []).
transpose(Matrix, [Row | Rows]) :-
    transpose_1st_col(Matrix, Row, RestMatrix),
    transpose(RestMatrix, Rows).
transpose_1st_col([], [], []).
transpose_1st_col([H | T] | Rows, [H | Hs], [T | Ts]) :-
    transpose_1st_col(Rows, Hs, Ts).

printMatrix([]) :- !.
printMatrix([Row | T]) :- stdio::write(Row), nl, printMatrix(T).

```

```

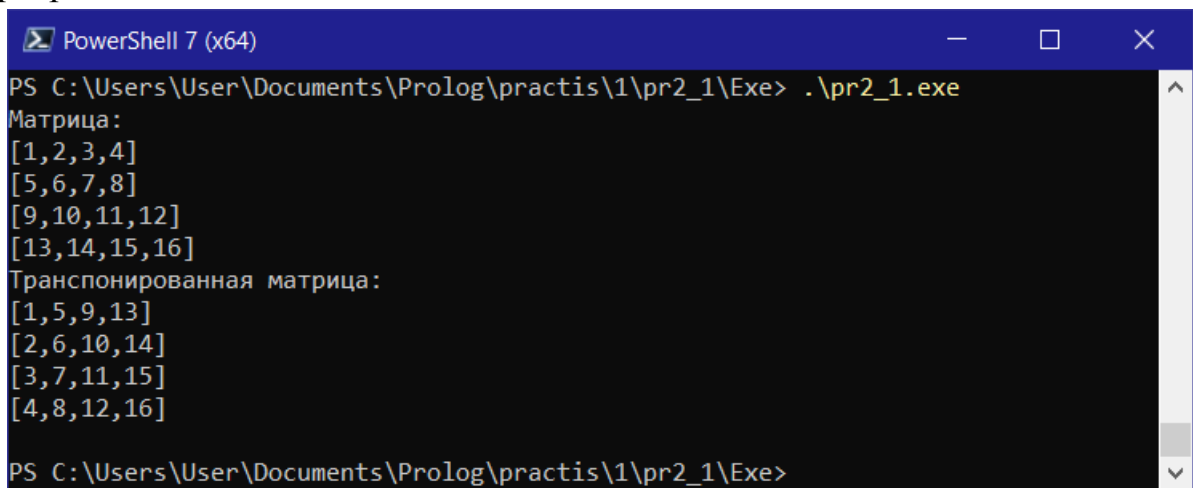
run() :-
    M = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]],
    stdio::write("Матрица:\n"), printMatrix(M),
    transpose(M, R),
    stdio::write("Транспонированная матрица:\n"), printMatrix(R),
    fail;
_ = stdio::readLine().

```

При тестировании предиката задаем матрицу $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$. При

транспонировании получим матрицу $\begin{bmatrix} 15 & 9 & 13 \\ 26 & 10 & 14 \\ 37 & 11 & 15 \\ 48 & 12 & 16 \end{bmatrix}$, что соответствует выводу

программы.



```

PowerShell 7 (x64)
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe> .\pr2_1.exe
Матрица:
[1,2,3,4]
[5,6,7,8]
[9,10,11,12]
[13,14,15,16]
Транспонированная матрица:
[1,5,9,13]
[2,6,10,14]
[3,7,11,15]
[4,8,12,16]
PS C:\Users\User\Documents\Prolog\practis\1\pr2_1\Exe>

```