```
dec lt : num -> list num;

dec lst : num -> list num;

dec cabeca : list num -> num;

dec cauda : list num -> num;

dec cauda2 : list num -> list num;

dec ultimo : list num -> num;

dec arranjo : list num -> list num;


--- lst 0 <= [];

--- lst n <= n :: lst (n - 1);


--- cabeca [] <= error ("Lista vazia");

--- cabeca (x :: xs) <= x;


--- cauda [] <= error ("Lista Vazia");


--- cauda2 [] <= error ("Lista Vazia");

--- cauda2 x <= [];

--- cauda2 (x :: xs) <= xs;


--- ultimo [] <= error ("Lista vazia");

--- ultimo ([x]) <= x;

--- ultimo (x :: xs) <= ultimo xs;


--- arranjo [] <= error ("Lista vazia");

--- arranjo ([x]) <= [];

--- arranjo (x :: xs) <= x :: arranjo xs;
```

```
dec faixa : num # num # num -> list num;

dec comp : list num # (num -> truval) -> list num;

dec listapot : num # list num -> list num;


--- faixa (i, f, p) <= if i > f then [] else i :: faixa (i + p, f, p);


--- comp ([], qualificador) <= [];

--- comp (x :: xs, qualificador) <= if qualificador x then x :: comp (xs, qualificador)
else comp (xs, qualificador);


--- listapot (n, []) <= [];

--- listapot (n, x :: xs) <= pow (x, n) :: listapot (n, xs);


uses base1, listas;

dec multiplo : num # num -> truval;

dec divisor : num -> list num;

dec listamult : list num # num -> list num;

dec divisor2 : num -> list num;

dec checa_primo : num -> truval;

dec lprimos : num -> list num;

dec ehPrimoAux : num # num -> truval;

dec ehPrimo : num -> truval;

dec lprimos2 : num -> list num;


--- multiplo (n, m) <= if n mod m = 0 then true else false;


--- divisor n <= comp (faixa (1, n, 1), lambda d => multiplo (n, d));
```

```
--- listamult ([], n) <= [];

--- listamult (x :: xs, n) <= if n mod x = 0 then x :: listamult (xs, n) else listamult (xs, n);


--- divisor2 n <= listamult (faixa (1, n, 1), n);


--- checa_primo 1 <= false;

--- checa_primo 2 <= true;

--- checa_primo n <= if tamanho (listamult (faixa (2, n - 1, 1), n)) > 0 then false else true;


--- lprimos n <= comp (faixa (1, n, 1), lambda x => checa_primo x);


--- ehPrimoAux (p, q) <= if pow (q, 2) > p then true else if p mod q = 0 then false else ehPrimoAux (p, q + 1);


--- ehPrimo p <= if p < 2 then false else ehPrimoAux (p, 2);


--- lprimos2 n <= if n < 2 then [] else if ehPrimo (n - 1) then lprimos2 (n - 1) <> [n - 1] else lprimos2 (n - 1);


dec x_pi : num;

dec acirc : num -> num;

dec soma : num # num -> num;


--- x_pi <= 3.14159;


--- acirc r <= x_pi * pow (r, 2);
```

```
--- soma (x, y) <= x + y;


uses recursao;
dec fatbase : num # num -> num;
dec fat2 : num -> num;
dec fibo2 : num # num # num -> num;
dec fib2 : num -> num;
dec par : num -> truval;
dec base2_1 : num -> num;
dec base2x : num # num -> num;
dec base2_2 : num -> num;
dec quantasVezes : num # num # num -> num;
dec qquantasVezes1 : num # num -> num;

--- fatbase (0, x) <= x;
--- fatbase (n, x) <= fatbase (n - 1, n * x);

--- fat2 n <= fatbase (n, 1);

--- fibo2 (0, anter, atual) <= anter;
--- fibo2 (1, anter, atual) <= atual;
--- fibo2 (2, anter, atual) <= atual + anter;
--- fibo2 (n, anter, atual) <= fibo2 (n - 1, atual, anter + atual);

--- fib2 n <= fibo2 (n, 0, 1);

--- par 0 <= true;
--- par n <= not (par (n - 1));
```

```
--- base2_1 0 <= 1;dec

--- base2_1 n <= 2 * base2_1 (n - 1);


--- base2x (0, x) <= x;

--- base2x (n, x) <= base2x (n - 1, x * 2);


--- base2_2 n <= base2x (n, 1);


--- quantasVezes (0, k, x) <= 0;

--- quantasVezes (n, k, x) <= if n = 0 then x else if n mod 10 = k then quantasVezes
(n, k div 10, x + 1) else quantasVezes (n, k div 10, x);


--- qquantasVezes1 (k, n) <= quantasVezes (k, n, 0);


uses base2;

dec membro : num # list num -> truval;

dec juncao : list num # list num -> list num;

dec unico : list num -> list num;

dec insira : num # list num -> list num;

dec classifica : list num -> list num;

dec uniao : list num # list num -> list num;

dec interseccao : list num # list num -> list num;

dec diferenca : list num # list num -> list num;

dec sub_lista : list num # list num -> truval;

dec igualdade : list num # list num -> truval;


--- membro (n, []) <= false;
```

```
--- membro (n, x :: xs) <= if n = x then true else membro (n, xs);


--- juncao ([], []) <= [];

--- juncao (a, []) <= a;

--- juncao ([], b) <= b;

--- juncao (a :: ax, b) <= a :: juncao (ax, b);


--- unico [] <= [];

--- unico (x :: xs) <= if membro (x, xs) then unico xs else x :: unico xs;


--- insira (n, []) <= [n];

--- insira (n, x :: xs) <= if n =< x then n :: x :: xs else x :: insira (n, xs);


--- classifica [] <= [];

--- classifica (x :: xs) <= insira (x, classifica xs);


--- uniao (a, b) <= classifica (unico (juncao (a, b)));


--- interseccao (a, []) <= [];

--- interseccao ([], b) <= [];

--- interseccao (a, x :: b) <= if membro (x, a) then x :: interseccao (a, b) else
interseccao (a, b);


--- diferenca (a, []) <= a;

--- diferenca ([], b) <= [];

--- diferenca (a :: ax, b) <= if membro (a, b) then diferenca (ax, b) else a :: diferenca
(ax, b);


--- sub_lista ([], ys) <= true;
```

```
--- sub_lista (x :: xs, ys) <= if membro (x, ys) then sub_lista (xs, ys) else false;

--- igualdade (a, b) <= sub_lista (a, b) and sub_lista (b, a);

uses base2, conjuntos;
dec b : list num;
dec a : list num;
dec c : list num;

--- b <= [1, 3, 5, 6, 7];

--- a <= [1, 3, 5, 7];

--- c <= [1, 2, 5, 6, 7, 8];

uses base2;
dec inverte : list num -> list num;
dec divisores : num -> list num;
dec produtolst : list num -> num;
dec faixa_primos : num # num -> list num;
dec multlstn : num # list num -> list num;
dec tabuada : num -> list num;
dec somalst : list num -> num;
dec amigos : num # num -> truval;
dec abundante : num -> truval;
dec div84 : num -> truval;
dec div84' : num -> truval;
dec multip_faixa : num # num -> num;
```

```
dec deficiente : num -> truval;

dec perfeito : num -> truval;

dec primeiros_enquanto : (num -> truval) # list num -> list num;

dec suspende_enquanto : (num -> truval) # list num -> list num;

dec primeiros : num # list num -> list num;

dec itera : (num -> truval) # num -> list num;

dec itera' : (num -> num) # num -> list num;


--- inverte [] <= [];

--- inverte (x :: xs) <= inverte xs <> [x];


--- divisores 0 <= [];

--- divisores n <= inverte (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x)));


--- produtolst [] <= 0;

--- produtolst ([x]) <= x;

--- produtolst (x :: xs) <= x * produtolst xs;


--- faixa_primos (0, 0) <= [];

--- faixa_primos (n, m) <= comp (faixa (n, m, 1), lambda x => checa_primo x);


--- multlstn (x, []) <= [];

--- multlstn (x, n :: ns) <= x * n :: multlstn (x, ns);


--- tabuada n <= multlstn (n, faixa (1, 10, 1));


--- somalst [] <= 0;

--- somalst (x :: xs) <= x + somalst xs;
```

```
--- amigos (x, y) <= somalst (divisores x) - x = y and somalst (divisores y) - y = x;

--- abundante n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n > n;

--- div84 n <= if n mod 8 = 4 then true else false;

--- div84' n <= n mod 8 = 4;

--- multip_faixa (x, y) <= if x > y then 0 else if x = y then y else x * multip_faixa (x + 1, y);

--- deficiente n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n < n;

--- perfeito n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n = n;

--- primeiros_enquanto (funcao, []) <= [];

--- primeiros_enquanto (funcao, x :: xs) <= if funcao x then x :: primeiros_enquanto (funcao, xs) else [];

--- suspende_enquanto (funcao, []) <= [];

--- suspende_enquanto (funcao, x :: xs) <= if funcao x then suspende_enquanto (funcao, xs) else x :: xs;

--- primeiros (0, _) <= [];

--- primeiros (_, []) <= [];

--- primeiros (n, x :: xs) <= x :: primeiros (n - 1, xs);
```

```
--- itera' (x, n) <= n :: itera' (x, x n);

dec primeiros : num # list num -> list num;
--- primeiros (0, []) <= [];
--- primeiros (n, []) <= [];

dec primeiros' : num # list num -> list num;
--- primeiros' (0, []) <= [];
--- primeiros' (n, []) <= [];
--- primeiros' (n, x :: xs) <= x :: primeiros' (n - 1, xs);

dec penultimo : list num -> num;
--- penultimo [] <= error ("lista vazia");
--- penultimo ([x]) <= error ("lista com um elemento");
--- penultimo (x :: y :: xs) <= penultimo (y :: xs);

dec maisDeTres : list num -> num;
--- maisDeTres [] <= 0;
--- maisDeTres (x :: xs) <= if x > 3 then x + maisDeTres xs else maisDeTres xs;

dec dolar : num # num -> num;
dec squad : num # num -> num;

--- dolar (real, valor) <= real / valor;

--- squad (a, b) <= a ^ 2 + b ^ 2;
```

```
uses base2;

dec inverte : list num -> list num;

dec divisores : num -> list num;

dec produtolist : list num -> num;

dec faixa_primo : num # num -> list num;

dec multlstn : num # list num -> list num;

dec tabuada : num -> list num;

dec tabuada' : num -> list num;

dec somalst : list num -> num;

dec amigos : num # num -> truval;

dec abundante : num -> truval;

dec abundante' : num -> truval;

dec div84 : num -> truval;

dec lstdiv84 : num -> list num;

dec multip_faixa : num # num -> num;

dec produto : list num -> num;

dec multipFaixa : num # num -> num;

dec deficiente : num -> truval;

dec deficiente' : num -> truval;

dec perfeito : num -> truval;

dec primeiros_enquanto : (num -> truval) # list num -> list num;

dec eh_par : num -> truval;

dec suspende_enquantos : (num -> truval) # list num -> list num;

dec suspende_enquanto' : (num -> truval) # list num -> list num;

dec primeiros : num # list num -> list num;

dec itera : (num -> num) # num -> list num;


--- inverte [] <= [];
```

```
--- inverte (x :: xs) <= inverte xs <> [x];


--- divisores 0 <= [];
--- divisores n <= inverte (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x)));


--- produtolist [] <= 0;
--- produtolist ([x]) <= x;
--- produtolist (x :: xs) <= x * produtolist xs;


--- faixa_primo (0, 0) <= [];
--- faixa_primo (0, 1) <= [1];
--- faixa_primo (n, m) <= comp (faixa (n, m, 1), lambda x => checa_primo x);


--- multlstn (0, [x]) <= [];
--- multlstn (n, x :: xs) <= x * n :: multlstn (x, xs);


--- tabuada n <= multlstn (n, []);


--- tabuada' n <= multlstn (n, faixa (1, 10, 1));


--- somalst [] <= 0;
--- somalst (x :: xs) <= x + somalst xs;


--- amigos (x, y) <= somalst (divisores x) - x = y and somalst (divisores y) - y = x;


--- abundante n <= somalst (divisores n) > n;
```

--- abundante' n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n > n;

--- div84 n <= if n mod 8 = 4 then true else false;

--- lstdiv84 n <= comp (faixa (1, n, 1), lambda x => div84 x);

--- produto [] <= 1;

--- produto (x :: xs) <= x * produto xs;

--- multipFaixa (x, y) <= if x > y then 0 else if x = y then y else x * multipFaixa (x + 1, y);

--- deficiente n <= somalst (divisores n) < n;

--- deficiente' n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n < n;

--- perfeito n <= somalst (comp (faixa (1, n + 1, 1), lambda x => multiplo (n, x))) - n = n;

--- primeiros_enquanto (funcao, []) <= [];

--- primeiros_enquanto (funcao, x :: xs) <= if funcao x then x :: primeiros_enquanto (funcao, xs) else primeiros_enquanto (funcao, xs);

--- eh_par x <= x mod 2 = 0;

--- suspende_enquantos (funcao, []) <= [];

--- suspende_enquantos (funcao, x :: xs) <= if funcao x then suspende_enquantos (funcao, xs) else x :: xs;

```
--- suspende_enquanto' (funcao, []) <= [];

--- suspende_enquanto' (funcao, x :: xs) <= if funcao x then suspende_enquanto' (funcao, xs) else x :: xs;


--- primeiros (0, _) <= [];

--- primeiros (_, []) <= [];

--- primeiros (n, x :: xs) <= x :: primeiros (n - 1, xs);


--- itera (x, y) <= y :: itera (x, x y);


uses listas;

dec somalst : list num -> num;

dec mult2 : list num -> list num;

dec duplique : list num -> list num;

dec intervalo : list num -> list num;

dec intervalo2 : num # num -> list num;

dec inverte : list num -> list num;

dec inverte' : list num -> list num;

dec poe_final : num # list num;

dec poe_final2 : num # list num -> list num;

dec poe_finalCerto : num # list num -> list num;

dec mult : list num -> list num;

dec mult3 : list num -> num;

dec invertBas : list num # list num -> list num;

dec inverte'' : list num -> list num;


--- somalst [] <= 0;
```

```
--- somalst (x :: xs) <= x + somalst xs;

--- mult2 [] <= [];
--- mult2 (x :: xs) <= x * 2 :: mult2 xs;

--- duplique [] <= [];
--- duplique (x :: xs) <= x :: duplique xs;
--- duplique (x :: xs) <= x :: x :: duplique xs;

--- intervalo2 (m, n) <= if m > n then [] else m :: intervalo2 (m + 1, n);

--- inverte [] <= [];
--- inverte (x :: xs) <= inverte xs <> [x];

--- inverte' [] <= [];
--- inverte' xs <= ultimo xs :: inverte' (arranjo xs);

--- poe_final2 (n, []) <= [];

--- poe_finalCerto (n, []) <= [n];
--- poe_finalCerto (n, x :: xs) <= x :: poe_finalCerto (n, xs);

--- mult [] <= [];

--- mult3 [] <= 1;
--- mult3 (x :: xs) <= x * mult3 xs;

--- invertBas ([], ys) <= ys;
```

```
--- invertBas (x :: xs, ys) <= invertBas (xs, x :: ys);

--- inverte'' xs <= invertBas (xs, []);

uses basico;
dec soma_com_10 : num -> num;
dec subtrai_2 : num -> num;
dec x_pi_em_5 : num -> num;

--- soma_com_10 x <= x + 10;

--- subtrai_2 x <= x - 2;

--- x_pi_em_5 x <= x * x_pi / 5;

uses array;
dec suspende : num # list num -> list num;
dec concatena : list num # list num -> list num;
dec remove : num # list num -> list num;
dec posicao : num # list num -> num;
dec rmvult : list num -> list num;
dec rmvult' : list num -> list num;
dec ate : num -> list num;
dec quadrados : num -> list num;
dec quadrados' : num -> list num;
dec quads_inv : num -> list num;

--- suspende (0, [xs]) <= [xs];
```

```
--- suspende (_, []) <= [];
--- suspende (n, x :: xs) <= suspende (n - 1, xs);

--- concatena ([], lst) <= lst;

--- remove (n, []) <= [];
--- remove (n, x :: xs) <= if n = x then xs else x :: remove (n, xs);

--- posicao (n, []) <= 0;
--- posicao (n, x :: xs) <= if n = x then 1 else 1 + posicao (n, xs);

--- rmvult [] <= error ("Lista vazia");
--- rmvult (x :: xs) <= arranjo xs;
--- rmvult ([x]) <= [];

--- rmvult' [] <= error ("Lista vazia");
--- rmvult' ([x]) <= [];

--- ate 0 <= [];
--- ate n <= ate (n - 1) <> [n - 1];

--- quadrados 0 <= [];
--- quadrados n <= quadrados (n - 1) <> [pow (n - 1, 2)];

--- quadrados' 0 <= [];
--- quadrados' n <= quadrados' (n - 1) <> [pow (n, 2)];

--- quads_inv 0 <= [];
```

```
--- quads_inv n <= [pow (n, 2)] <> quads_inv (n - 1);


dec contaDigitos : num -> num;
dec SomaDigitos : num -> num;
dec mediaDigitos : num -> num;
dec collaz : num -> num;
dec calcN : num -> num;
dec exp : num # num -> num;
dec somaFaixa : num # num -> num;
dec dec2bin : num -> num;
dec sucessor : num -> num;
dec antecessor : num -> num;
dec minhaSoma : num # num -> num;
dec produto : num # num -> num;
dec bin2dec : num -> num;
dec binon : num # num -> num;


--- contaDigitos 0 <= 0;
--- contaDigitos n <= 1 + contaDigitos (n div 10);


--- SomaDigitos 0 <= 0;
--- SomaDigitos n <= n mod 10 + SomaDigitos (n div 10);


--- mediaDigitos n <= SomaDigitos n / contaDigitos n;


--- collaz 1 <= 1;
--- collaz n <= if abs n mod 2 = 0 then collaz (abs n div 2) else collaz (abs n * 3 + 1);
```

```
--- calcN 0 <= 0;

--- calcN n <= 1.5 + calcN (n - 1) / 2;


--- exp (1, 1) <= 1;

--- exp (x, n) <= x * exp (x, n - 1);


--- somaFaixa (m, n) <= if n < m then 0 else if m = n then m else n + somaFaixa (m, n
- 1);


--- dec2bin 0 <= 0;

--- dec2bin n <= if n = 1 then 1 else n;

--- dec2bin n <= 10 * dec2bin (n div 2) + n mod 2;


--- sucessor n <= n + 1;


--- antecessor n <= n - 1;


--- minhaSoma (0, b) <= b;

--- minhaSoma (a, 0) <= a;

--- minhaSoma (0, 0) <= 0;

--- minhaSoma (a, b) <= minhaSoma (sucessor a, antecessor b);


--- produto (0, 0) <= 0;

--- produto (0, b) <= 0;

--- produto (a, 0) <= 0;

--- produto (1, b) <= b;

--- produto (a, 1) <= a;

--- produto (a, b) <= a + produto (a, b - 1);
```

```
--- bin2dec 0 <= 0;

--- bin2dec n <= 2 * bin2dec (n div 10) + n mod 10;


--- binon (n, 0) <= 1;

--- binon (n, k) <= (n - 1) / (k - 1) + (n - 1) / k;

--- binon (n, k) <= if n < k then error ("n < k") else if k = 0 then 1 else if k = n then 1
else binon (n - 1, k - 1) + binon (n - 1, k);


uses recursao;

dec prodIntervalo1 : num # num -> num;

dec resto : num # num -> num;

dec somaCubos : num -> num;

dec contaPares : num -> num;

dec somaDigitos : num -> num;

dec somaDigitos2 : num -> num;

dec primeiroDigito : num -> num;

dec primeiroDigito2 : num -> num;

dec zeros : num -> num;

dec somaQuadrados : num -> num;


--- prodIntervalo1 (m, n) <= if m > n then 1 else m * prodIntervalo1 (m + 1, n);


--- resto (p, q) <= p - q * (p / q);


--- somaCubos 0 <= 0;

--- somaCubos n <= pow (n, 3) + somaCubos (n - 1);
```

```
--- contaPares 0 <= 1;
--- contaPares 1 <= 1;
--- contaPares n <= if n mod 2 = 0 then contaPares (n - 1) + 1 else contaPares (n - 1);

--- somaDigitos 0 <= 0;
--- somaDigitos n <= n mod 10 + somaDigitos (n div 10);

--- somaDigitos2 n <= if n = 0 then 0 else n mod 10 + somaDigitos2 (n div 10);

--- primeiroDigito n <= n mod 10;

--- primeiroDigito2 n <= n div 10;
--- primeiroDigito2 0 <= 0;

--- zeros 0 <= 1;
--- zeros n <= if n < 10 then 0 else if n mod 10 = 0 then 1 + zeros (n div 10) else zeros
(n div 10);

--- somaQuadrados 0 <= 0;
--- somaQuadrados n <= n ^ 2 + somaQuadrados (n - 1);

dec ehPotenciaDeDois : num -> truval;
dec mediaDigitos : num -> num;
dec contaDigitos : num -> num;
dec somaDigitos : num -> num;
dec collatz : num -> num;
dec calc : num -> num;
dec exp : num # num -> num;
```

```
dec somaFaixa : num # num -> num;

dec dec2bin : num -> num;

dec sucessor : num -> num;

dec antecessor : num -> num;

dec minhaSoma : num # num -> num;

dec produto : num # num -> num;

dec bin2dec : num -> num;

dec binon : num # num -> num;


--- ehPotenciaDeDois 1 <= true;

--- ehPotenciaDeDois n <= if n > 1 and n mod 2 = 0 then ehPotenciaDeDois (n div 2) else false;


--- mediaDigitos n <= somaDigitos n / contaDigitos n;


--- contaDigitos 0 <= 0;

--- contaDigitos n <= 1 + contaDigitos (n div 10);


--- somaDigitos 0 <= 0;

--- somaDigitos n <= n mod 10 + somaDigitos (n div 10);


--- collatz 1 <= 1;

--- collatz n <= if abs n mod 2 = 0 then collatz (abs n div 2) else collatz (abs n * 3 + 1);


--- calc 0 <= 0;

--- calc n <= 1.5 + calc (n - 1) / 2;

--- calc n <= (1.5 + calc (n - 1)) / 2;
```

```
--- exp (x, 0) <= 0;

--- exp (x, n) <= x * exp (x, n - 1);


--- somaFaixa (m, n) <= if n < m then 0 else if n = m then m else n + somaFaixa (m, n
- 1);


--- dec2bin 0 <= 0;

--- dec2bin 1 <= 1;

--- dec2bin n <= 10 * dec2bin (n div 2) + n mod 2;


--- sucessor n <= n + 1;


--- antecessor n <= n - 1;


--- minhaSoma (0, b) <= b;

--- minhaSoma (a, 0) <= a;

--- minhaSoma (0, 0) <= 0;

--- minhaSoma (a, b) <= minhaSoma (sucessor a, antecessor b);


--- produto (x, 0) <= 0;

--- produto (0, y) <= 0;

--- produto (x, y) <= x + produto (x, y - 1);


--- bin2dec 0 <= 0;

--- bin2dec n <= 2 * bin2dec (n div 10) + n mod 10;


--- binon (n, 0) <= 1;

--- binon (n, k) <= if n < k then error ("n < k") else if k = 0 then 1 else if k = n then 1
else binon (n - 1, k - 1) + binon (n - 1, k);
```

```
dec fibo : num # num -> list num;

dec lstfib : num -> list num;

dec primeiros4 : num # list num -> list num;


--- fibo (n, m) <= n :: fibo (m, n + m);

--- fibo (n, m) <= n :: fibo (m, n + m);


--- lstfib n <= primeiros4 (n, fibo (0, 1));


--- primeiros4 (0, _) <= [];

--- primeiros4 (_, []) <= [];

--- primeiros4 (n, x :: xs) <= x :: primeiros4 (n - 1, xs);


uses basico;

dec par : num -> truval;

dec par' : num -> truval;

dec impat : num -> truval;

dec impar'' : num -> truval;

dec valor : num -> num;


--- par n <= if n mod 2 = 0 then true else false;


--- par' n <= n mod 2 = 0;


--- impat n <= n mod 2 /= 0;


--- valor 0 <= 0;
```

```
--- valor 1 <= 2;
--- valor n <= if n > 1 and n < 9 then n * 5 else n / 5;

uses fluxo;
dec negativo : num -> num;
dec classif : num # num -> num # num;
dec div84 : num -> truval;
dec extenso : num -> list char;
dec calculadora : num # num # num -> num;

--- negativo 0 <= 0;
--- negativo n <= if n < 0 then n else 0 - n;

--- div84 n <= if n mod 8 = 4 then true else false;

--- extenso 0 <= "zero";
--- extenso 1 <= "um";
--- extenso 2 <= "dois";
--- extenso 3 <= "tres";
--- extenso 4 <= "quatro";
--- extenso 5 <= "cinco";
--- extenso 6 <= "seis";
--- extenso 7 <= "sete";
--- extenso 8 <= "oito";
--- extenso 9 <= "nove";
--- extenso _ <= "Valor invalido";

--- calculadora (x, y, 0) <= x + y;
```

```
--- calculadora (x, y, 1) <= x - y;
--- calculadora (x, y, 10) <= x * y;
--- calculadora (x, y, 11) <= x / y;
--- calculadora (x, y, _) <= error ("valor invalido");

uses array;
dec elem2 : list num -> num;
dec tamanho : list num -> num;
dec elempnt : list num -> num;

--- elem2 [] <= error ("Lista vazia");
--- elem2 ([x]) <= error ("Poucos elementos");
--- elem2 (a :: b :: bs) <= b;

--- tamanho [] <= 0;
--- tamanho (x :: xs) <= 1 + tamanho xs;

--- elempnt [] <= error ("Lista Vazia");
--- elempnt ([x]) <= error ("Poucos numeros");
--- elempnt (x :: xs) <= if tamanho xs = 1 then x else elempnt xs;

dec x_pi : num;
dec area_circ : num -> num;
dec qsoma : num # num -> num;
dec x_e : num;
dec x_pi_em_e : num -> num;
dec suc : num -> num;
dec c2f : num -> num;
```

```
dec f2c : num -> num;

dec c2k : num -> num;

dec k2c : num -> num;


--- x_pi <= 3.14159;


--- area_circ r <= x_pi * r ^ 2;


--- qsoma (a, b) <= a ^ 2 + b ^ 2;


--- x_e <= 2.71828;


--- x_pi_em_e x <= x * x_pi / x_e;


--- suc x <= x + 1;


--- c2f c <= c * 9 / 5 + 32;


--- f2c f <= (f - 32) * 5 / 9;


--- c2k c <= c + 273.15;


--- k2c k <= k - 273.15;


dec x_pi : num;

dec acirc : num -> num;

dec qsoma : num # num -> num;

dec x_e : num;
```

```
dec x_pi_em_x : num -> num;

dec suc : num -> num;

dec c2f : num -> num;

dec c2k : num -> num;

dec metro2centimetro : num -> num;

dec saoiguais : num # num -> truval;

dec ant : num -> num;

dec cubo : num -> num;

dec k2f : num -> num;

dec f2k : num -> num;

dec imc : num # num -> num;

dec produto : num # num -> num;

dec eq1grau : num # num -> num;

dec area_ret : num # num -> num;


--- x_pi <= 3.14159;


--- acirc r <= x_pi * pow (r, 2);


--- qsoma (x, y) <= pow (x, 2) + pow (y, 2);


--- x_e <= 2.71828;


--- x_pi_em_x a <= a * x_pi / x_e;


--- suc x <= x + 1;


--- c2f f <= (f - 32) * 5 / 9;
```

```
--- c2k c <= c + 273.15;

--- metro2centimetro c <= c * 100;

--- saoiguais (a, b) <= if a = b then true else false;

--- ant x <= x - 1;

--- cubo c <= c ^ 3;

--- k2f k <= (k - 273.15) * 9 / 5 + 32;

--- f2k f <= (f - 32) * 5 / 9 + 273.15;

--- imc (p, a) <= p / pow (a, 2);

--- produto (x, y) <= x * y;

--- eq1grau (a, b) <= b / a;

--- area_ret (lado1, lado2) <= lado1 * lado2;

uses man;
dec x_pi : num;
--- x_pi <= 3.14159;

dec acirc : num -> num;
```

```
--- acirc r <= x_pi * pow (r, 2);

dec soma : num # num -> num;
--- soma (x, y) <= x + y;

dec suc : num -> num;
--- suc res <= res+1;

dec ant : num -> num;
--- ant res <= res-1;

dec qsoma : num # num -> num;
--- qsoma (x,y) <= (x+y) ^ 2;

dec cubo : num -> num;
--- cubo res <= pow(res,3);

dec produto : num # num -> num;
--- produto (x,y) <= x*y;

dec imc : num # num -> num;
--- imc (p,a) <= p/(a^2);

dec dolar : num # num -> num;
--- dolar (real,valor) <= real / valor;

dec squad : num # num -> num;
--- squad (a,b) <= (a^2)+(b^2);
```

```
dec par : num -> truval;

--- par n <= if n mod 2 = 0 then true else false;

uses fluxo;
dec fat1 : num -> num;
dec fib1 : num -> num;
dec potencia : num # num -> num;

--- fat1 0 <= 1;
--- fat1 n <= n * fat1 (n - 1);

--- fib1 0 <= 0;
--- fib1 1 <= 1;
--- fib1 n <= fib1 (n - 1) + fib1 (n - 2);

--- potencia (n, 0) <= 1;
--- potencia (b, e) <= b * potencia (b, e - 1);

uses recursao;
dec fbs : num # num # num -> num;
dec fib2 : num -> num;
dec par : num -> truval;
dec base2_1 : num -> num;
dec base2x : num # num -> num;
dec base2_2 : num -> num;
dec quantasVezes : num # num -> num;
```

```
dec qvbase : num # num # num -> num;

dec quantasVezes2 : num # num -> num;


--- fbs (0, anter, atual) <= anter;

--- fbs (1, anter, atual) <= atual;

--- fbs (2, anter, atual) <= atual + anter;

--- fbs (n, anter, atual) <= fbs (n - 1, atual, anter + atual);


--- fib2 n <= fbs (n, 0, 1);


--- par 0 <= true;

--- par n <= not (par (n - 1));


--- base2_1 0 <= 1;

--- base2_1 n <= 2 * base2_1 (n - 1);


--- base2x (0, x) <= x;

--- base2x (n, x) <= base2x (n - 1, x * 2);


--- base2_2 n <= base2x (n, 1);


--- qvbase (k, n, i) <= if n = 0 then i else if n mod 10 = k then qvbase (k, n div 10, i + 1)
else qvbase (k, n div 10, i);


--- quantasVezes2 (k, n) <= qvbase (k, n, 0);


fat1 : num -> num;

fat1 0 <= 1;
```

```
fat1 n <= n * fat1(n-1);


fib1 : num -> num;
fib1 0 <= 0;
fib1 1 <= 1;
fib1 n <= fib1 (n-1) + fib1(n-2);


uses base2, conjuntos, basico, fluxo;
dec mapa : list num # (num -> num) -> list num;
dec filtro : list num # (num -> truval) -> list num;
dec reducao : list num # (num # num -> num) # num -> num;
dec compacta : list num # list num -> list (num # num);
dec oposto : list num -> list num;
dec dcp_base : list num # list num # list (num # num) -> list num # list num;
dec descompacta : list (num # num) -> list num # list num;


--- mapa ([], funcao) <= [];
--- mapa (x :: xs, funcao) <= funcao x :: mapa (xs, funcao);


--- filtro ([], funcao) <= [];
--- filtro (x :: xs, funcao) <= if funcao x then x :: filtro (xs, funcao) else filtro (xs,
funcao);


--- reducao ([], funcao, n) <= n;
--- reducao (x :: xs, funcao, n) <= funcao (x, reducao (xs, funcao, n));


--- compacta ([], b) <= [];
--- compacta (a, []) <= [];
```

```
--- compacta (x :: xs, y :: ys) <= (x, y) :: compacta (xs, ys);


--- oposto [] <= [];
--- oposto (x :: xs) <= oposto xs <> [x];


--- dcp_base (xs, ys, []) <= (oposto xs, oposto ys);
--- dcp_base (xs, ys, (x, y) :: zs) <= dcp_base (x :: xs, y :: ys, zs);


--- descompacta [] <= ([], []);
--- descompacta xs <= dcp_base ([], [], xs);


uses array, listas;


dec intervalo : num # num -> list num;
--- intervalo (m, n) <= if m > n then [] else m :: intervalo (m + 1, n);


dec mult2 : list num -> list num;
--- mult2 [] <= [];
--- mult2 (x :: xs) <= x * 2 :: mult2 xs;


dec duplique : list num -> list num;
--- duplique [] <= [];
--- duplique (x :: xs) <= x :: x :: duplique xs;


dec somalista : list num -> num;
--- somalista [] <= error ("Lista Vazia");
--- somalista (x :: xs) <= x + somalista xs;
```

```
dec inverte : list alpha -> list alpha;

--- inverte [] <= [];

--- inverte (x :: xs) <= inverte xs <> [x];


dec invert' : list num -> list num;

--- invert' [] <= [];

--- invert' xs <= ultimo xs :: invert' (arranjo xs);



dec poe_final : num # list num -> list num;

--- poe_final (n, []) <= [n];

--- poe_final (n, x :: xs) <= x :: poe_final (n, xs);


dec mult3 : list num -> num;

--- mult3 [] <= 1;

--- mult3 (x :: xs) <= x * mult3 xs;


uses listas, array, recursao;


dec faixa : num # num # num -> list num;

--- faixa (i, f, p) <= if i > f then [] else i :: faixa (i + p, f, p);


dec comp : list num # (num -> truval) -> list num;

--- comp ([], qualificador) <= [];

--- comp (x :: xs, qualificador) <= if qualificador x then x :: comp (xs, qualificador)
else comp (xs, qualificador);


//Função anônima.
```

```
comp ([1,2,3,4], \ x => x mod 2 = 0);
comp(faixa(1,7,1), \ x => par (x)); //Outro exemplo.



dec listapot : num # list num -> list num;
--- listapot (n, []) <= [];
--- listapot (n, x :: xs) <= pow (x, n) :: listapot (n, xs);
```


```
dec listapot : num # list num -> list num;
--- listapot (n, []) <= [];
--- listapot (n, x :: xs) <= pow (x, n) :: listapot (n, xs);
```


```
dec par : num -> truval;
dec impar : num -> truval;
dec valor : num -> num;
dec min : num # num -> num;
dec max : num # num -> num;
dec sinal : num # num -> num;
dec abc : num -> num;

--- par n <= if n mod 2 = 0 then true else false;

--- impar n <= if n mod 2 /= 0 then true else false;

--- valor 0 <= 0;
```

```
--- valor 1 <= 2;
--- valor n <= if n > 1 and n < 9 then n * 5 else n / 5;

--- min (a, b) <= if a < b then a else b;

--- max (a, b) <= if a > b then a else b;

--- sinal (x, y) <= if x < y then 0 - 1 else if x > y then 1 else 0;

--- abc 0 <= 1;
--- abc 1 <= 2;
--- abc 2 <= 4;
--- abc x <= if x > 2 then 9 else 0;

dec par : num -> truval;
dec impar : num -> truval;
dec valor : num -> num;
dec min : num # num -> num;
dec max : num # num -> num;
dec sinal : num # num -> num;
dec abc : num -> num;

--- par n <= if n mod 2 = 0 then true else false;

--- impar n <= if n mod 2 /= 0 then true else false;

--- valor 0 <= 0;
--- valor 1 <= 2;
```

```
--- valor n <= if n > 1 and n < 9 then n * 5 else n / 5;


--- min (a, b) <= if a < b then a else b;


--- max (a, b) <= if a > b then a else b;


--- sinal (x, y) <= if x < y then 0 - 1 else if x > y then 1 else 0;


--- abc 0 <= 1;

--- abc 1 <= 2;

--- abc 2 <= 4;

--- abc x <= if x > 2 then 9 else 0;
```