FEUP-L.EIC024-2024/2025-1S

Início	quinta-feira, 9 de janeiro de 2025 às 14:35
Estado	Prova submetida
Data de submissão:	quinta-feira, 9 de janeiro de 2025 às 16:37
Tempo gasto	2 horas 2 minutos
Nota	**20,00** de um máximo de 20,00 (**100**%)

Informação

PT

Read these instructions carefully before you start the test.

- This is an open-book test; you can use the materials available on the local computer and Moodle's course page, but no printed materials are allowed.
- You can use the resources available on the computer, such as SICStus Prolog.
- **The presence of electronic and/or communication devices is strictly forbidden.**
- The maximum duration of the test is stated below.
- The score of each question is stated in the test, totaling 20 points.
- A wrong answer in a multiple-choice question with five options implies a penalty of 25% of the question's value.
- A wrong answer in a True/False question implies a penalty of 50% of the question's value.
- Fraud attempts will be punished by the annulment of the test for all intervenients.

- **You can use predicates requested in previous questions even if you haven't implemented them.**
- **If you implement auxiliary predicates, you must include them in the answers to all questions where they are used.**
- **Do not include the answer to previous questions in the current one.**
- **Do not copy any code provided in the questions into the answer text box.**
- **Document your code and use intuitive variable names so as to clarify code interpretation.**
- **Pay attention to syntactic errors (such as forgetting the '.' after each rule/fact).**

Be mindful of the available time.

Good work!

Informação

Consider the following knowledge base regarding books and authors.

PT

```
%author(AuthorID, Name, YearOfBirth, CountryOfBirth).
author(1,       'John Grisham',         1955,   'USA').
author(2,       'Wilbur Smith',         1933,   'Zambia').
author(3,       'Stephen King',         1947,   'USA').
author(4,       'Michael Crichton',     1942,   'USA').

%book(Title, AuthorID, YearOfRelease, Pages, Genres).
book('The Firm',                1,      1991,   432,    ['Legal thriller']).
book('The Client',              1,      1993,   422,    ['Legal thriller']).
book('The Runaway Jury',        1,      1996,   414,    ['Legal thriller']).
book('The Exchange',            1,      2023,   338,    ['Legal thriller']).

book('Carrie',                  3,      1974,   199,    ['Horror']).
book('The Shining',             3,      1977,   447,    ['Gothic novel', 'Horror', 'Psychological horror']).
book('Under the Dome',          3,      2009,   1074,   ['Science fiction', 'Political']).
book('Doctor Sleep',            3,      2013,   531,    ['Horror', 'Gothic', 'Dark fantasy']).

book('Jurassic Park',           4,      1990,   399,    ['Science fiction']).
book('Prey',                    4,      2002,   502,    ['Science fiction', 'Techno-thriller', 'Horror',
'Nanopunk']).
book('Next',                    4,      2006,   528,    ['Science fiction', 'Techno-thriller', 'Satire']).
```

Pergunta 1

Pontuou 1,500 de 1,500

Implement ***book_author(?Title, ?Author)***, which associates a book title with the name of its author.

PT

```
% book_author(?Title, ?Author)
book_author(Title, Author) :-
    book(Title, AuthorId, _, _
    author(AuthorId, Author, _
```

```
book_author(Title, Author):-
   author(AuthID, Author, _YoB, _CoB),
   book(Title, AuthID, _YoP, _Pages, _Genres).

% This is similar to creating a View in SQL joining two tables (CREATE VIEW book_author AS SELECT title, name FROM
author JOIN book on authorID)
```

Pergunta 2

Pontuou 1,500 de 1,500

Implement ***multi_genre_book(?Title)***, which unifies ***Title*** with the title of a book that has multiple genres. Example:

PT

```
| ?- multi_genre_book(Title).
Title = 'The Shining' ? ;
Title = 'Under the Dome' ?
yes
| ?-
```

```
% multi_genre_book(?Title)
multi_genre_book(Title) :-
    book(Title, _, _, _, Genres),
    length(Genres, NGenres),
    NGenres > 1.
```

```
multi_genre_book(Title):-
       book(Title, _AuthID, _Year, _Pages, [_One, _Two | _Rest]).  % Has at least two genres

% Alternative
multi_genre_book(Title):-
    book(Title, _AuthID, _Year, _Pages, Genres),
       length(Genres, Len),                      % length is a built-in predicate
       Len > 1.

% Another alternative
multi_genre_book(Title):-
       book(Title, _AuthID, _Year, _Pages, Genres),
       member(A, Genres),                        % member is a built-in predicate
       member(B, Genres),                        % member is a built-in predicate
       A \= B.
```

Pergunta 3

Pontuou 1,500 de 1,500

Implement ***shared_genres(?Title1, ?Title2, -CommonGenres)***, which receives two book titles as arguments and returns on the third argument a list containing the genres that are common to both books. Any order of the shared genres is valid.
Example:

PT

```
| ?- shared_genres('Prey', 'Next', Shared).
Shared = ['Science fiction','Techno-thriller'] ? ;
no
```

```
% shared_genres(?Title1, ?Title
shared_genres(Title1, Title2, C
    book(Title1, _, _, _, Genre
    book(Title2, _, _, _, Genre
    shared_genres_aux(Genres1,

% shared_genres_aux(+Genres1, +
shared_genres_aux([], _Genres2,
shared_genres_aux([Genre | Tail
    member(Genre, Genres2), !,
```

```
shared_genres(Title1, Title2, CommonGenres):-
        book(Title1, _ID1, _Year1, _Pages1, Genres1),
        book(Title2, _ID2, _Year2, _Pages2, Genres2),
        common_elements(Genres1, Genres2, CommonGenres).

% Determine common elements between two lists
common_elements([], _L, []).
common_elements([H|T], L, [H|R]):-
        member(H, L), !,
        common_elements(T, L, R).
common_elements([_|T], L, R):-
        common_elements(T, L, R).

% Note that there is no guarantee regarding the order or position of the elements in the lists
% Note that the cut is necessary to avoid obtaining additional (wrong) solutions via backtracking
```

Pergunta 4

Pontuou 1,500 de 1,500

PT

The Jaccard coefficient, also known as intersection over union (IoU), is a similarity measurement between two sets, determined by the division between the intersection (number of common elements between the two sets) and the union (total number of different elements in both sets). Implement the ***similarity(?Title1, ?Title2, ?Similarity)*** predicate, which determines the Jaccard coefficient between the two books received as first two arguments, considering the genres of each book as the measure of similarity. Example:

```
| ?- similarity('The Firm', 'The Client', Sim).
Sim = 1.0 ? ;
no
| ?- similarity('Prey', 'Next', Sim).
Sim = 0.4 ?
yes
```

```
% similarity(?Title1, ?Title2,
similarity(Title1, Title2, Simi
    book(Title1, _, _, _, Genre
    book(Title2, _, _, _, Genre
    shared_genres(Title1, Title
    length(Genres1, N1),
    length(Genres2, N2),
    length(Intersection, NInt),
    NUnion is N1 + N2 - NInt,
    Similarity is NInt / NUnion
```

```
similarity(Title1, Title2, Similarity):-
    shared_genres(Title1, Title2, Intersection),
    book(Title1, _ID1, _Year1, _Pages1, Genres1),
    book(Title2, _ID2, _Year2, _Pages2, Genres2),
    union(Genres1, Genres2, Union),
    length(Intersection, LI),
    length(Union, LU),
    Similarity is LI / LU.

union(Set1, Set2, UnionSet):-
    append(Set1, Set2, All),    % append is a built-in predicate
    sort(All, UnionSet).        % sort is a built-in predicate (sorts and removes duplicates)

% Note the use of the predicate from the previous question (shared_genres)
% The length of the Union set could be determined by length(Genres1, L1), length(Genres2, L2), LU is L1+L2-LI.
```

Pergunta 5

Pontuou 0,500 de 0,500

We want to be able to write queries and code in the format *Book has N pages*. Example:

PT

```
'The Firm' has 432 pages.
```

What definition of operators makes more sense to allow writing code in this format?

a.
```
:-op(700, xfx, has).
:-op(600, xf, pages).
```
✔

b.
```
:-op(700, xfx, has).
:-op(600, fx, pages).
```

c.
```
:-op(700, yfy, has).
:-op(600, xf, pages).
```

d.
```
:-op(700, xfx, has).
:-op(700, xf, pages).
```

e.
```
:-op(700, yfy, has).
:-op(600, fx, pages).
```

Resposta correta:

```
:-op(700, xfx, has).
:-op(600, xf, pages).
```

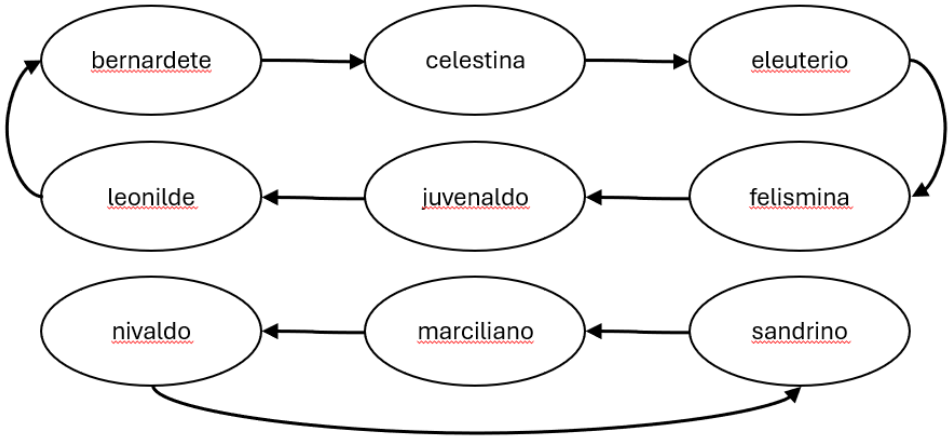
Informação

The Passionate Fans of Literature (PFL) Book Club instituted a Secret Santa for Christmas. In a draw of luck, each member of the club was assigned another member for whom to buy a gift. Being a book club, everyone bought a book, and the full information was registered after the PFL Christmas Dinner in the following database:

PT

```
% gives_gift_to(Giver, Gift, Receiver)
gives_gift_to(bernardete, 'The Exchange', celestina).
gives_gift_to(celestina, 'The Brethren', eleuterio).
gives_gift_to(eleuterio, 'The Summons', felismina).
gives_gift_to(felismina, 'River God', juvenaldo).
gives_gift_to(juvenaldo, 'Seventh Scroll', leonilde).
gives_gift_to(leonilde, 'Sunbird', bernardete).
gives_gift_to(marciliano, 'Those in Peril', nivaldo).
gives_gift_to(nivaldo, 'Vicious Circle', sandrino).
gives_gift_to(sandrino, 'Predator', marciliano).
```

The following image represents this information graphically.



During the dinner, starting with the acting club president, each member gave a small speech, disclosed to whom they bought a gift, and handed it to the receiver. The receiver then goes on stage and continues the process, opening his/her gift, making a small speech, and giving the gift to the next receiver. When the initial speaker goes on stage again, closing the circle, another member is selected to begin the process again.

Informação

In the following questions, you can use multiple solution predicates (findall, setof, and bagof) as well as any SICStus library.

PT

Pergunta 6

Pontuou 1,500 de 1,500

Implement ***circle_size(+Person, ?Size)***, which unifies the second argument with the number of people who form the circle of gifts that includes the person received as first argument. Examples:

PT

```
| ?- circle_size(bernardete, Size).
Size = 6 ? ;
no
| ?- circle_size(marciliano, Size).
Size = 3 ?
yes
```

```
:- use_module(library(lists)).

% circle_size(+Person, ?Size)
circle_size(Person, Size) :-
    circle_bfs([Person], [Perso
    length(Circle, Size).

% circle_bfs(+Queue, +Acc, -Vis
circle_bfs([], Acc, Visited) :-
circle_bfs([Person | QueueTail]
```

```
circle_size(Person, Size):-
    collect([Person], People),
    length(People, Size).

collect( [H|T], People):-
    gives_gift_to(H, _, N),
    \+ member(N, [H|T]), !,
    collect( [N,H|T], People).
collect(People, People).

% Note that the cut is necessary to avoid additional (wrong) answers via backtracking
% This exercise is similar to a depth-first search (PL5, slide 17) and some exercises from the TP classes (e.g., P05,
exercise 4)
```

Pergunta 7

Pontuou 2,000 de 2,000

Implement ***largest_circle(?People)***, which unifies ***People*** with the list of people belonging to the largest circle in the book club. The order of the people in the list is not important. In case there is more than one circle with the largest dimension, the predicate should succeed more than once. The order of the results is not important. Example:

PT

```
| ?- largest_circle(People).
People = [bernardete,celestina,eleuterio,felismina,juvenaldo,leonilde] ? ;
no
```

```
 :- use_module(library(lists)).

 % largest_circle(?People)
 largest_circle(People) :-
     circles([], Circles),
     findall(Len-Circle, (
         member(Circle, Circles),
         length(Circle, Len)
     ), LenCircles),
     max_member(MaxLen-_, LenCir
```

```
:-use_module(library(lists)).

largest_circle(People):-
    all_people(Everyone),
    setof(Size-Person-Sorted, Persons^(member(Person, Everyone), collect([Person], Persons), sort(Persons, Sorted),
length(Sorted, Size)), Triples),
    last(Triples, MaxSize-_-_),
    setof(Persons,  P^member(MaxSize-P-Persons, Triples), LargestGroups),
    member(People, LargestGroups).

all_people(List):-
    findall(X, (gives_gift_to(X, _, _) ; gives_gift_to(_, _, X)), Temp),
    sort(Temp, List).

% The collect/2 predicate from the previous question is reused here
% Note that there may be several circles (groups of people) with the same size (collected in LargestGroups).
```

Pergunta 8

Pontuou 0,500 de 0,500

What does this query do?

PT

```
? - T =.. L, append([_,_],[X|_],L), var(X).
```

a. It tests if the third argument of a term T is uninstanciated.

b. It tests if the second argument of a term T is an atom.

c. It tests if the second argument of a term T is uninstanciated. ✔

d. It tests if the first argument of a term T is uninstanciated.

e. It tests if the third argument of a term T is an atom.

Resposta correta:
It tests if the second argument of a term T is uninstanciated.

Pergunta 9

Pontuou 0,500 de 0,500

Consider the existence of a predicate predX/3 and the following queries:

PT

```
A) findall(X, predX(1, X, _), L).
B) setof(X, predX(1, X, _), L).
C) setof(X, Y^predX(1, X, Y), L).
D) bagof(X, predX(1, X, _), L).
E) bagof(X, Y^predX(1, X, Y), L).
```

Which of the previous queries are equivalent?

a. Only A and E are equivalent. ✔

b. Only C and E are equivalent.

c. Only B and D are equivalent.

d. A, B and D are equivalent.

e. Only A and C are equivalent.

Resposta correta:
Only A and E are equivalent.

Pergunta 10

Pontuou 1,500 de 1,500

Implement ***dec2bin(+Dec, -BinList, +N)***, which converts a non-negative integer number ***Dec*** into a list of bits representing that number, using exactly ***N*** bits. If the number of bits is insufficient to represent the number, the predicate should fail. If the number to convert is negative, the predicate should fail. Examples:

PT

```
| ?- dec2bin(7, List, 8).
List = [0,0,0,0,0,1,1,1] ? ;
no
| ?- dec2bin(10, List, 8).
List = [0,0,0,0,1,0,1,0] ?
yes
| ?- dec2bin(1234, List, 8).
no
```

```
% dec2bin(+Dec, -BinList, +N)
dec2bin(0, [], 0).
dec2bin(Dec, [0 | BinTail], N)
    Dec >= 0,
    N > 0,
    NextN is N - 1,
    Dec < 2^NextN,
    dec2bin(Dec, BinTail, NextN
dec2bin(Dec, [1 | BinTail], N)
    Dec >= 0
```

```
dec2bin(Dec, List, N):-
       Dec >= 0,
       dec2bin(Dec, [], List, N).

dec2bin(0, List, List, 0):- !.
dec2bin(Dec, Acc, List, N):-
       N > 0,
       Bit is Dec mod 2,
       Next is Dec div 2,
       N1 is N - 1,
       dec2bin(Next, [Bit|Acc], List, N1).

% Note that if the decimal number cannot fit in the given number of bits, the predicate will fail
```

Pergunta 11

Pontuou 1,500 de 1,500

Implement ***initialize(+DecNumber, +Bits, +Padding, -List)***, which receives a decimal number and number of bits in which to represent it, as well as ***Padding*** - the number of zeroes to place on each side of the resulting binary representation - returning in ***List*** the resulting list of bits. Example:

PT

```
| ?- initialize(12, 4, 2, List).
List = [0,0,1,1,0,0,0,0] ? ;
no
```

Note that 12 is represented as [1,1,0,0] when using 4 bits; this representation is then surrounded by padding lists of two zeroes on both sides.

```
:- use_module(library(between)).

% initialize(+DecNumber, +Bits, +
initialize(DecNumber, Bits, Paddi
    dec2bin(DecNumber, Bin, Bits)
    findall(0, between(1, Padding
    append(Bin, PaddingList, Temp
    append(PaddingList, TempList,
```

```
initialize(Dec, N, Padd, List):-
    dec2bin(Dec, Mid, N),
    dec2bin(0, Side, Padd),
    append([Side, Mid, Side], List).

% Note the use of the dec2bin predicate to obtain both the binary representation of the decimal number and the padding
lists
% The append/2 predicate (from the lists library) appends a lists of lists
```

Pergunta 12

Pontuou 1,500 de 1,500

Implement ***print_generation(+List)***, which prints to the terminal a text representation of a list of bits, representing 0 as a dot ('.'), 1 as a capital M ('M'), and separating each byte (set of 8 bits) with a pipe ('|'). Examples:

PT

```
| ?- print_generation([0,0,0,0,1,0,1,0]).
|....M.M.|
yes
| ?- print_generation([0,0,0,0,1,0,1,0,0,0,1,1]).
|....M.M.|..MM
yes
| ?- print_generation([0,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0]).
|....M.M.|..M.M.M.|
yes
```

```
% print_generation(+List)
print_generation(List) :-
    length(Byte, 8),
    append(Byte, Rest, List), !
    write('|'),
    print_byte(Byte),
    print_generation(Rest).
print_generation(List) :-
    write('|'),
    print_byte(List).
```

```
print_line(0, []):- !,
    write('|'), nl.
print_line(_, []):- !, nl.
print_line(0, Bits):- !,
    write('|'),
    print_line(8, Bits).
print_line(N, [Bit|Bits]):-
    N1 is N-1,
    translate(Bit, Char),
    put_char(Char),
    print_line(N1, Bits).

translate(0, '.').
translate(1, 'M').

print_generation(L):-
    write('|'),
    print_line(8, L).

% This exercise is similar to printing a board with side borders (as in the practical assignment)
```

Pergunta 13

Pontuou 1,500 de 1,500

PT

The one-dimensional version of Conway's Game of Life can be played using a list of cells that can be either dead or alive. On each generation, each cell's state can change depending on its own state and the state of its neighboring cells according to the defined transition rules.

The rules for updating each cell from one generation to the next consider the state of the left neighbor, the cell itself, and the right neighbor. For each of the eight possible combinations, the ***rule(?Config, ?State)*** predicate gives the resulting cell state, where the first argument is a three-bit configuration represented as a compound term Left-Self-Right, and the second argument is the resulting state bit.

For instance, the fact *rule(1-0-0, 1).* represents the configuration where the left neighbor is active and the cell itself and its right neighbor are both dead; the state of the cell in the next generation is active. With eight possible combinations of bits, the eight resulting states can be represented as a number from 0 to 255.

Implement ***update_rule(+Rule)***, which receives a number between 0 and 255, and changes the knowledge base so that exactly eight *rule/2* facts exist. This predicate should always succeed, except in case it receives a value outside the expected range of values. Example:

```
| ?- update_rule(10).
yes
| ?- listing(rule/2).
rule(0-0-0, 0).
rule(0-0-1, 1).
rule(0-1-0, 0).
rule(0-1-1, 1).
rule(1-0-0, 0).
rule(1-0-1, 0).
rule(1-1-0, 0).
rule(1-1-1, 0).

yes
```

Note that 10 is represented as [0,0,0,0,1,0,1,0] when using 8 bits. The most significant bit corresponds to the 1-1-1 combination of bits. The least significant bit corresponds to the 0-0-0 combination of bits.

```
| ?- update_rule(210).
yes
| ?- listing(rule/2).
rule(0-0-0, 0).
rule(0-0-1, 1).
rule(0-1-0, 0).
rule(0-1-1, 0).
rule(1-0-0, 1).
rule(1-0-1, 0).
rule(1-1-0, 1).
rule(1-1-1, 1).

yes
```

```
:- use_module(library(lists)).
:- use_module(library(between)).

% update_rule(+Rule)
update_rule(Rule) :- Rule < 0, !.
update_rule(Rule) :- Rule > 255, !.
update_rule(_Rule) :-
    retractall( rule(_, _) ),
    fail.
update_rule(Rule) :-
```

```
update_rule(N):-
    \+ (dec2bin(N, Bits, 8)), !, fail.
update_rule(N):-
    dec2bin(N, Bits, 8),
    abolish(rule/2),
    between(0, 1, FirstBit),
    between(0, 1, SecondBit),
    between(0, 1, ThirdBit),
    Index is 8 - (FirstBit * 4 + SecondBit * 2 + ThirdBit),
    nth1(Index, Bits, Bit),
    assert( rule(FirstBit-SecondBit-ThirdBit, Bit) ),
    fail.
update_rule(_).

% The first clause ensures a fail in case the value is not valid
% The last clause ensures the predicate succeeds otherwise (i.e., when the rules are updated)
% Note the need to remove existing clauses before asserting the new ones
```

Pergunta 14

Pontuou 1,500 de 1,500

Implement ***next_gen(+Previous, -Next)***, which receives a list of binary values and computes the next generation, applying the existing rules to each position. Missing neighbors (for the first and last elements) are assumed to be zeroes. Example:

PT

```
| ?- update_rule(210), next_gen([0,0,1,0,0,0,1,0], NextGen).
NextGen = [0,1,0,1,0,1,0,1] ? ;
no
```

```
% next_gen(+Previous, -Next)
next_gen([X1, X2 | PrevTail], [
    rule(0-X1-X2, Y1),
    next_gen_aux([X1, X2 | Prev

% next_gen_aux(+Previous, -Next
next_gen_aux([X1, X2], [Y1]) :-
next_gen_aux([X1, X2, X3 | Prev
    rule(X1-X2-X3, Y1),
    next_gen_aux([X2, X3 | Prev
```

```
next_gen(Gen0, Gen1):-
    apply_rules(0, Gen0, Gen1).

apply_rules(Left, [Self], [New]):-
    rule(Left-Self-0, New).
apply_rules(Left, [Self, Right | Rest], [New | Tail]):-
    rule(Left-Self-Right, New),
    apply_rules(Self, [Right|Rest], Tail).
```

Pergunta 15

Pontuou 1,500 de 1,500

PT

Implement ***play(+DecNumber, +Bits, +Padding, +Rule, +N)***, which receives the input values for the *initialize/4* predicate (***DecNumber***, ***Bits***, and ***Padding***), the input value for the *update_rule/1* predicate (***Rule***), and ***N***, the number of generations to simulate. This predicate should orchestrate the calls to existing predicates and print the first ***N*** generations to the terminal. Note that a call to *play/5* with ***N*** = 1 results in printing the initial generation only. Example:

```
| ?- play(128, 16, 8, 210, 12).
|........|........|M.......|........|
|........|.......M|.M......|........|
|........|......M.|..M.....|........|
|........|.....M.M|.M.M....|........|
|........|....M...|....M...|........|
|........|...M.M..|...M.M..|........|
|........|..M...M.|..M...M.|........|
|........|.M.M.M.M|.M.M.M.M|........|
|........|M.......|........|M.......|
|.......M|.M......|.......M|.M......|
|......M.|..M.....|......M.|..M.....|
|.....M.M|.M.M....|.....M.M|.M.M....|
true ?
yes
```

```
% play(+DecNumber, +Bits, +Padd
play(DecNumber, Bits, Padding,
    initialize(DecNumber, Bits,
    update_rule(Rule),
    game_loop(State, N).

% game_loop(+State, +N)
game_loop(_State, 0).
game_loop(State, N) :-
    N > 0.
```

```
play(Init, N, Padd, Rule, Gens):-
    initialize(Init, N, Padd, Gen0),
    update_rule(Rule),
    play_gens(Gens, Gen0).

play_gens(1, Gen):- !,
    print_generation(Gen).
play_gens(N, Gen0):-
    N1 is N -1,
    print_generation(Gen0),
    next_gen(Gen0, Gen1),
    play_gens(N1, Gen1).
```

◀ MT1 - Grades

MT2 - Grades ▶