

Informação

No concurso FEUPGotTalent, cada estudante pode participar mostrando as suas habilidades num qualquer tema, académico ou extra-curricular. Os interessados inscrevem-se, dando o número de estudante, idade e o nome da sua atuação:

```
%participant(Id, Age, Performance)
participant(1234, 17, 'Pé coxinho').
participant(3423, 21, 'Programar com os pés').
participant(3788, 20, 'Sing a Bit').
participant(4865, 22, 'Pontes de esparguete').
participant(8937, 19, 'Pontes de pen-drives').
participant(2564, 20, 'Moodle hack').
```

As atuações são apreciadas por um júri de **E** elementos.

Ao longo da atuação (que tem um máximo de 120 segundos), se um elemento do júri achar que o participante não deve passar à próxima fase, carrega num botão. Ficam registados os tempos em que cada elemento do júri carregou no botão. Se não carregou, ficam registados 120 segundos.

```
%performance(Id, Times)
performance(1234, [120, 120, 120, 120]).
performance(3423, [32, 120, 45, 120]).
performance(3788, [110, 2, 6, 43]).
performance(4865, [120, 120, 110, 120]).
performance(8937, [97, 101, 105, 110]).
```

Passam à próxima fase os **N** participantes que mais se aguentaram em palco, somados os tempos de cada elemento do júri, desde que pelo menos um dos elementos do júri não tenha carregado no botão.

Responda às perguntas 1 a 5 **SEM** utilizar predicados de obtenção de múltiplas soluções (findall, setof e bagof).

Pergunta 1

Resposta guardada

Pontuação 1,00

Implemente o predicado ***madeItThrough(+Participant)***, que sucede se *Participant* é um participante que já atuou e em cuja atuação pelo menos um elemento do júri não carregou no botão.

| ?- madeItThrough(1234).
yes

| ?- madeItThrough(2564).
no

| ?- madeItThrough(3788).
no



```
madeItThrough(Participant) :-  
    performance(Participant, List),  
    member(X, List),  
    X = 120.
```

Pergunta 2

Resposta guardada

Pontuação 1,50

Implemente o predicado ***juriTimes(+Participants, +JuriMember, -Times, -Total)***, que devolve em *Times* o tempo de atuação de cada participante na lista *Participants* (pela mesma ordem) até que o júri número *JuriMember* (de 1 a E) carregou no botão, e em *Total* a soma desses tempos.

| ?- juriTimes([1234,3423,3788,4865,8937],1,Times,Total).
Times = [120,32,110,120,97],
Total = 479

| ?- juriTimes([1234,3423,3788,4865,8937],2,Times,Total).
Times = [120,120,2,120,101],
Total = 463



```
juriTimes([], _, [], 0).  
juriTimes([H|T], JuriMember, [H1|T1], Total) :-  
    juriTimes(T, JuriMember, T1, TotalAux),  
    performance(H, TimesList),  
    nth1(JuriMember, TimesList, H1),  
    Total is (TotalAux + H1).
```

Pergunta 3

Resposta guardada

Pontuação 1,00

Implemente o predicado ***patientJuri(+JuriMember)*** que sucede se o júri *JuriMember* já se absteve de carregar no botão pelo menos por duas vezes.

| ?- patientJuri(3).
no

| ?- patientJuri(4).
yes



```
patientJuri(JuriMember) :-  
    performance(X, TimeList),  
    performance(Y, TimeList2),  
    X \= Y,  
    nth0(JuriMember, TimeList, Time),  
    nth0(JuriMember, TimeList2, Time2),  
    Time == 120,  
    Time2 == 120.
```

Pergunta 4

Resposta guardada

Pontuação 1,50

Implemente o predicado ***bestParticipant(+P1, +P2, -P)*** que unifica *P* com o melhor dos dois participantes *P1* e *P2*. O melhor participante é aquele que tem uma maior soma de tempos na sua atuação (independentemente de estar ou não em condições de passar à próxima fase). Se ambos tiverem o mesmo tempo total, o predicado deve falhar.

| ?- bestParticipant(3423,1234,Z).
Z = 1234

| ?- bestParticipant(1234,1234,Z).
no



```
bestParticipant(P1, P2, P) :-  
    performance(P1, Times1),  
    performance(P2, Times2),  
    sumlist(Times1, Total1),  
    sumlist(Times2, Total2),  
    Total1 > Total2,  
    P is P1.
```

```
bestParticipant(P1, P2, P) :-  
    performance(P1, Times1),  
    performance(P2, Times2),  
    sumlist(Times1, Total1),  
    sumlist(Times2, Total2),  
    Total1 < Total2,  
    P is P2.
```

Pergunta 5

Resposta guardada

Pontuação 1,00

Implemente o predicado ***allPerfs***, que imprime na consola os números dos participantes que já atuaram, juntamente com o nome da sua atuação e lista de tempos.

| ?- allPerfs.

1234:Pé coxinho:[120,120,120,120]

3423:Programar com os pés:[32,120,45,120]

3788:Sing a Bit:[110,2,6,43]

4865:Pontes de esparguete:[120,120,110,120]

8937:Pontes de pen-drives:[97,101,105,110]

yes



```
allPerfs :-  
    performance(P, Times),  
    participant(P, _, Performance),  
    write(P), write(':'), write(Performance), write(':'), write(Times), nl,  
    fail.  
  
allPerfs.
```

Informação

Nas perguntas seguintes pode fazer uso de predicados de obtenção de múltiplas soluções (findall, setof e bagof).

Pergunta 6

Resposta guardada

Pontuação 1,00

Implemente o predicado ***nSuccessfulParticipants(-T)*** que determina quantos participantes não tiveram qualquer clique no botão durante a sua atuação.

| ?- nSuccessfulParticipants(T).
T = 1



```
nSuccessfulParticipants(T) :-  
    setof(_Participant, (performance(_Participant, TimesList),  
no_click_of_button(TimesList)), Total),  
    length(Total,T).
```

```
not_member(_,[]).  
not_member(X,[H|T]) :-  
    X \= H,  
    not_member(X, T).
```

```
no_click_of_button([]).  
no_click_of_button([Head|Tail]):-  
    Head == 120,  
    no_click_of_button(Tail).
```

Por responder Pontuação 1,50

$$L = [1234-[1,2,3,4], 3423-[2,4], 3788-[], 4865-[1,2,4], 8937-[]]$$

A \underline{A} ▼
 B
 I

U
 $\frac{\textcircled{\scriptsize S}}$
 X_2
 x^2

A ▼
 ▼

☰ ☷

☰ ☷ ☷

🔗 🔌

🖼️

⏪ ⏩

√α Ω

📊

T_x

↶ ↷

👁️/ 👂

<> 🔄

Por responder Pontuação 1,50

O seguinte predicado permite obter participantes, suas atuações e tempos totais, que estejam em condições de passar à próxima fase: para um participante poder passar, tem de haver pelo menos um elemento do júri que não tenha carregado no botão durante a sua atuação.


```
eligibleOutcome(Id,Perf,TT) :-
    performance(Id,Times),
    madeItThrough(Id),
    participant(Id,_,Perf),
    sumlist(Times,TT).
```

```
| ?- nextPhase(2,P).  
P = [480-1234-'Pé coxinho',470-4865-'Pontes de esparguete']
```

```
| ?- nextPhase(3,P).  
P = [480-1234-'Pé coxinho',470-4865-'Pontes de esparguete',317-3423-'Prog  
ramar com os pés']
```

```
| ?- nextPhase(4,P).
no
```

A ▼

B

I


U


$\frac{\text{S}}{\text{S}}$


X_2


X^2


A ▼


 ▼





























\sqrt{a}


Ω





T_x














Pergunta 9

Por responder

Pontuação 1,00

Explique o que faz o predicado ***predX/3*** apresentado abaixo. Indique ainda se o *cut* utilizado é verde ou vermelho, justificando a sua resposta.


```

predX(Q,[R|Rs],[P|Ps]) :-
    participant(R,I,P), I=<Q, !,
    predX(Q,Rs,Ps).
predX(Q,[R|Rs],Ps) :-
    participant(R,I,_), I>Q,
    predX(Q,Rs,Ps).
predX(_,[],[]).

```

A ▼
 B
 I

U
~~S~~
 X_2
 X^2

A ▼
 ▼



☰


$\frac{1}{2}$
 $\frac{3}{3}$

☰

☰

☰







☰


☰

\sqrt{a}
 Ω

 T_x

<>


Informação

Dado um número N , pretende-se determinar uma sequência de $2 \cdot N$ números que contenha, para todo o $k \in [1, N]$, uma sub-sequência $S_k = k, \dots, k$ começada e terminada com o número k e com k outros números de permeio. Por exemplo, a sequência $[2, 3, 1, 2, 1, 3]$ cumpre os requisitos: os 1s têm 1 número no meio, os 2s têm 2 números no meio, e os 3s têm 3 números no meio. A sequência $[2, 3, 4, 2, 1, 3, 1, 4]$ também cumpre. No entanto, alguns valores de N não têm solução possível.

Pergunta 10

Por responder Pontuação 1,00

Explique o que faz o seguinte predicado:

```
impoe(X,L):-
    length(Mid,X),
    append(L1,[X| ],L), append( ,[X|Mid],L1).
```

A ▼

B

I


U


~~S~~


X_2


X^2


A ▼


 ▼





























\sqrt{a}


Ω





T_x














Pergunta 11

Por responder

Pontuação 1,50

Tirando partido do predicado anterior, implemente o predicado ***langford(+N,-L)***, em que ***N*** é um inteiro dado e ***L*** será uma sequência de $2*N$ números conforme indicado atrás. (Nota: Langford foi o matemático escocês que propôs este problema.)

```
| ?- langford(3,L).
```

$L = [3, 1, 2, 1, 3, 2] ? ;$

$L = [2, 3, 1, 2, 1, 3] ? ;$

no

```
| ?- langford(4,L).
```

$L = [4, 1, 3, 1, 2, 4, 3, 2]$? ;

L = [2,3,4,2,1,3,1,4] ? ;

no

```
| ?- langford(5,L).
```

no

