

## Painel do utilizador

## Programação em Lógica

## Painel da UC

## Participantes

## Pauta

## Descarregar ficheiros

# Programação em Lógica

unidades curriculares

Programação em Lógica

Provas

Mini-Teste 1 Modelo

pergunta

estudante pode participar mostrando as suas habilidades num qualquer tema, académico ou extra-curricular. Os interessados inscrevem-se e o nome da sua atuação:

(exemplo) 'Jogar futebol', 'Dançar', 'Fazer malabarismo com os pés', 'Fazer magia', 'Fazer bit', 'Fazer esparguete', 'Fazer pen-drives', 'Fazer haka'.

O tempo de cada elemento é de 120 segundos.

Se o júri achar que o participante não deve passar à próxima fase, carrega num botão. Ficam registados os tempos de cada elemento do júri carregou no botão. Se não carregou, ficam registados 120 segundos.

[120,120],  
[120,120],  
[120,120],  
[120,120],  
[120,120],  
[120,120].

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 soluções múltiplas (findall, setof e bagof), e SEM usar qualquer biblioteca do SICStus.**

## Pergunta 1

Não modificada desde a última tentativa Pontuação 1,00 Destacar pergunta

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 carregou no botão.

```
| ?- madeItThrough(1234).
yes
| ?- madeItThrough(2564).
no
| ?- madeItThrough(3788).
no
```

madeItThrough(Participant) :-  
performance(Participant, List),  
loopmadeItThrough(List).  
  
loopmadeItThrough([]) :- false.  
  
loopmadeItThrough([Head | \_]) :-  
Head == 120.  
  
loopmadeItThrough([\_| Tail]) :-  
!,  
loopmadeItThrough(Tail).

## Pergunta 2

Não modificada desde a última tentativa Pontuação 1,50 Destacar pergunta

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(ListParticipants, JuriMember, Times, Total) :-  

loopJuriTimes(ListParticipants, JuriMember, [], Times, Total).

sum([], OldTotal, OldTotal).

sum([Head | Tail], OldTotal, Total) :-  

NewTotal is OldTotal + Head,  

sum(Tail, NewTotal, Total).

loopJuriTimes([], _, OldTimes, OldTimes, Total) :-  

sum(OldTimes, 0, Total).

```

### Pergunta 3

Não modificada desde a última tentativa Pontuação 1,00 Destacar pergunta

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) :-  

getParticipants([], ListOfParticipants),  

noPushButtonJuri(ListOfParticipants, JuriMember, 0, N),  

N >= 2.

noPushButtonJuri([], _OldN, OldN).

noPushButtonJuri([Participant | Tail], JuriMember, OldN, N) :-  

performance(Participant, TimeList),  

findTimeJuri(TimeList, JuriMember, TimeJuri),  

(TimeJuri == 120,  

N1 is OldN + 1);  

(N1 is OldN),
N1 is OldN,

```

### Pergunta 4

Resposta guardada Pontuação 1,50 Destacar pergunta

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, ListP1),  

sum(ListP1, 0, SumP1),  

performance(P2, ListP2),  

sum(ListP2, 0, SumP2),  

((SumP2 > SumP1, P is P2);  

(SumP1 > SumP2, P is P1)).

```

### Pergunta 5

Resposta guardada Pontuação 1,00 Destacar pergunta

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 coixinho:[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** :-  
 getParticipants([], ListOfParticipants),  
 loopAllPerfs(ListOfParticipants).

loopAllPerfs([]).

loopAllPerfs([ID | Tail]) :-  
 write(ID), write(','),  
 participant(ID, \_TypePerformance),  
 write(TypePerformance), write(','),  
 performance(ID, ListTime),  
 write(ListTime), write('\n').

## Informação

Destacar pergunta

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

## Pergunta 6

Resposta guardada Pontuação 1,00 Destacar pergunta

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

| ?- nSuccessfulParticipants(T).  
 T = 1

noButtonParticipants([]).

noButtonParticipants([120 | Tail]) :-  
 noButtonParticipants(Tail).

noButtonParticipants(\_) :- fail.

nSuccessfulParticipants(T):-  
 findall(ID, (performance(ID, ListTimes), noButtonParticipants(ListTimes)), ListIDs),  
 length(ListIDs, LenListIDs),  
 T is LenListIDs.

## Pergunta 7

Resposta guardada Pontuação 1,50 Destacar pergunta

Implemente o predicado **juriFans(juriFansList)**, que obtém uma lista contendo, para cada participante, a lista dos elementos do júri que não carregaram no botão ao longo da sua atuação.

| ?- juriFans(L).  
 L = [1234-[1.2,3.4],3423-[2.4],3788-[],4865-[1.2,4],8937-[]]

juriFans(JuriFansList) :-  
 findall(ID-NewList, (performance(ID, ListTime), getListJuri(ListTime, 0, [], NewList)), JuriFansList).

getListJuri([], \_, OldList, OldList).

getListJuri([ListSeconds | Tail], N, OldList, NewList):-  
 ListSeconds == 120,  
 N1 is N + 1,  
 append(OldList, [N1], List),  
 !,  
 getListJuri(Tail, N1, List, NewList).

getListJuri(\_ | Tail), N, OldList, NewList):-

## Pergunta 8

Resposta guardada Pontuação 1,50 Destacar pergunta

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.

| :- use\_module(library(lists)).

```
eligibleOutcome(Id,Perf,TT) :-  
    performance(Id,Times),  
    madeltThrough(Id),  
    participant(Id,_Perf),  
    sumlist(Times,TT).
```

Fazendo uso deste predicado, implemente o predicado ***nextPhase(+N, -Participants)***, que obtém a lista com os tempos totais, números e atuações dos *N* melhores participantes, que passarão portanto à próxima fase. Se não houver pelo menos *N* participantes a passar, o predicado deve falhar.

```
| ?- nextPhase(2,P).  
P = [480-1234-'Pé coixinho',470-4865-'Pontes de esparguete']  
| ?- nextPhase(3,P).  
P = [480-1234-'Pé coixinho',470-4865-'Pontes de esparguete',317-3423-'Programar com os pés']  
| ?- nextPhase(4,P).  
no
```

```
nextPhase(N,P) :-  
    setof(TT-Id-Perf, eligibleOutcome(Id, Perf, TT), ListEligible),  
    reverse(ListEligible, NewListEligible),  
    length(NewListEligible, LenList),  
    LenList >= N,  
    findall(E, (nth1(Index, NewListEligible, E), Index =< N), P).
```

## Pergunta 9

Resposta guardada Pontuação 1,00 Destacar pergunta

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(_,[],[]).
```

O predicado apresentado retorna no Terceiro Argumento o Nome das Atuações feitas por individuos com idade igual ou inferior ao primeiro argumento do predicado *predX*.

O cut é verde, pois influencia apenas a eficiência do programa (corta árvores de exploração), não tendo qualquer influência no resultado do programa.

[Página seguinte](#)

[TP2 - Enunciado](#)

Ir para...

SICStus Prolog ▶