Universidade Federal de Uberlândia
Faculdade de Computação
Programação Lógica (GSI010) — Prof. Dr. Paulo H. R. Gabriel

Segunda Lista de Exercícios

Observação: Faça todas as questões sem o auxílio do interpretador Prolog. Após resolvê-las, use algum interpretador para verificar suas respostas; caso apresente algum erro, procure entendê-lo e refaça a questão. Utilize o interpretador SWISH, online, disponível no link: <https://swish.swi-prolog.org/>.

1. O programa a seguir associa a cada pessoa seu esporte preferido.

```
joga(ana,volei).
joga(bia,tenis).
joga(ivo,basquete).
joga(eva,volei).
joga(leo,tenis).
```

   Suponha que desejamos consultar esse programa para encontrar um parceiro `P` para jogar com Leo. Então, podemos realizar essa consulta de duas formas:

   (a) `?- joga(P,X), joga(leo,X), P\=leo.`

   (b) `?- joga(leo,X), joga(P,X), P\=leo.`

   Desenhe as árvores de busca construídas pelo sistema ao responder cada uma dessas consultas. Qual consulta é mais eficiente e por quê?

2. Codifique um programa contendo as informações da tabela abaixo e faça as consultas indicadas a seguir:

Nome	Sexo	Idade	Altura	Peso
Ana	fem	23	1.55	56.0
Bia	fem	19	1.71	61.3
Ivo	masc	22	1.80	70.5
Lia	fem	17	1.85	57.3
Eva	fem	28	1.75	68.7
Ary	masc	25	1.72	68.9

   (a) Quais são as mulheres com mais de 20 anos de idade?

   (b) Quem tem pelo menos 1.70m de altura e menos de 65kg?

   (c) Quais são os possíveis casais onde o homem é mais alto que a mulher?

3. O peso ideal para uma modelo é no máximo $62.1 * \text{ altura } - 44.7$. Além disso, para ser modelo, uma mulher precisa ter mais que 1.70m de altura e menos de 25 anos de idade. Com base nessas informações, e considerando a tabela do exercício anterior, defina um predicado capaz de recuperar apenas os nomes das mulheres que podem ser modelos.

4. A potenciação (x^y) pode ser resolvida de maneira recursiva da seguinte maneira:

$$x^y = \begin{cases} 1, \text{ se } y = 0 \\ x \cdot x^{y-1}, \text{ caso contrário} \end{cases}$$

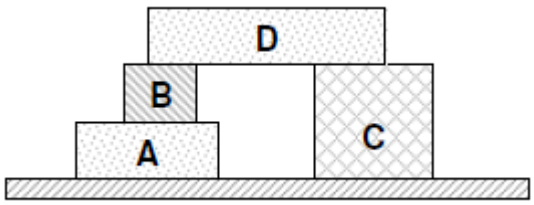
Implemente o predicado recursivo `potencia(Base, Expoente, Resultado)` que, dado um valor para a base x e um para o expoente y, calcule o valor o valor de x^y.

5. A função de Ackermann é definida para dois valores inteiros e não negativos, m e n, da seguinte forma:

$$ack(m,n) = \begin{cases} n+1, \text{ se } y = 0 \\ ack(m-1,1), \text{ se } m > 0 \text{ e } n = 0 \\ ack(m-1, ack(m, n-1)), \text{ se } m > 0 \text{ e } n > 0 \end{cases}$$

Implemente o predicado recursivo `ack(M,N,A)` que calcula o valor da função de Ackermann para dois valores inteiros. **Observação:** Teste esse predicado com valores pequenos de m e n (na ordem de 4, até 5), pois essa função gera números muito grandes em muito poucas interações!

6. Considere que um conjunto de blocos foi empilhado em uma mesa, conforme mostrado na figura a seguir:



   (a) Crie uma base de fatos utilizando o predicado `sobre(Bloco1, Bloco2)` que é verdadeiro se `Bloco1` está posicionado diretamente sobre o `Bloco2` (ou seja, ambos estão em contato direto). Por exemplo, `sobre(b,a)` é um fato válido, enquanto `sobre(d,a)` não é válido.

   (b) Crie uma regra recursiva `acima(X,Y)` que mostre todos os blocos (`X`) que estão acima de um determinado bloco `Y`. Exemplo de execução:

```
?- acima(X,a).
X = b ;
X = d
```

7. O predicado `num` classifica números em três categorias: positivos, nulo e negativos. Esse predicado, da maneira como está definido, realiza retrocesso (*backtracking*) desnecessário. Explique por que isso acontece e, em seguida, utilize cortes para eliminar esse retrocesso.

```
num(N,positivo) :- N > 0.
num(0,nulo).
num(N,negativo) :- N < 0.
```

8. Considere o seguinte programa:

```
d(0).
d(1).
b([A,B,C]) :- d(A), d(B), d(C).
```

   (a) Mostre o resultado da seguinte consulta: `?- b(N).` Desenhe a árvore de busca para essa consulta.

   (b) Substitua a terceira cláusula por `bin([A,B,C]) :- d(A), !, d(B), d(C).` O que muda no resultado da consulta do item anterior? Justifique, construindo uma nova árvore de busca.

   Observação: Esse programa utiliza, como saída, uma *lista*. Veremos essa estrutura nas próximas aulas.