Lista de Linguagens de Programação – 20

Nome: _____ Matrícula: _____

Os exercícios 1–6 usam os predicados abaixo:

```
parent(kim,holly).
parent(margaret,kim).
parent(margaret,kent).
parent(esther,margaret).
parent(herbert,margaret).
parent(herbert,jean).
greatgrandparent(GGP,GGC) :-
  parent(GGP,GP), parent(GP,P), parent(P,GGC).
```

1. Defina um predicado `mother`, tal que `mother(X, Y)` seja verdade se `X` for mãe de `Y`.

2. Defina um predicado `father`, tal que `father(X, Y)` seja verdade se `X` for o pai de `Y`.

3. Defina um predicado `sister`, tal que `sister(X, Y)` seja verdade se `X` for uma irmã de `Y`. Note que ninguém pode ser sua própria irmã.

4. Defina um predicado `grandson`, tal que `grandson(X,Y)` seja verdade se `X` for um neto de `Y`.

5. Defina um predicado `firstCousin`, tal que `firstCousin(X, Y)` seja verdade se `X` for um primo primeiro de `Y`. Lembre-se que uma pessoa não é primo primeiro dela mesma. Tampouco irmãos são primos.

6. Defina um predicado `descendant`, tal que `descendant(X, Y)` seja verdade se `X` for um descendente de `Y`.

7. Defina um predicado `third`, tal que `third(X, Y)` seja verdade se `Y` for o terceiro elemento da lista `X`. Isto pode ser expresso como um *fato*. Note que este predicado deverá produzir um erro se a lista possuir menos que três elementos.

8. Defina um predicado `firstPair`, tal que `firstPair(X)` seja verdade se `X` for uma lista com pelo menos dois elemento, sendo o primeiro elemento o mesmo que o segundo. Este predicado também pode ser expresso como um fato.

9. Defina um predicado `del3`, tal que `del3(X, Y)` seja verdade se `Y` for uma lista igual a `X`, mas com o terceiro elemento removido. Este predicado também pode ser expresso como um fato. Para facilitar as coisas, ele pode gerar erro se `X` possuir menos que três elementos.

10. Defina um predicado `dupList`, tal que `dupList(X, Y)` seja verdade se `X` for a mesma lista que `Y`, mas com cada elemento de `Y` repetido uma vez. Por exemplo, se `X` for a lista `[1, 2, 3]`, então `Y` deverá ser a lista `[1, 1, 2, 2, 3, 3]`. Se `X` for `[]`, então `Y` será `[]` também. Verifique se seu predicado funcione em ambas as direções. Isto é, a busca `dupList(X, [1, 1, 2, 2, 3, 3]` deve retornar a o resultado `X = [1, 2, 3]`.

11. Defina um predicado `isDuped`, tal que `isDuped(Y)` seja verdade se `Y` for uma lista igual a lista `Y` do exercício anterior. Ou seja, o predicado é verdade se `Y` for uma lista com o primeiro elemento igual ao segundo, o terceiro igual ao quarto, e assim por diante.

12. Defina um predicado `oddSize`, tal que `oddSize(X)` seja verdade se `X` for uma lista de tamanho ímpar. Não é necessário computar o tamanho da lista. Na verdade, não é preciso realizar nenhuma aritmética para resolver este exercício.

13. Defina um predicado `evenSize`, tal que `evenSize(X)` seja verdade se `X` for uma lista de tamanho par.

14. Defina um predicado `prefix`, tal que `prefix(X, Y)` seja verdade se `X` for um prefixo da lista `Y`. Isto é, cada elemento de `X` deve unificar com cada elemento correspondente de `Y`, mas `Y` pode conter mais elementos que `X`. Seu predicado deve funcionar quando `X` não estiver instanciado. Por exemplo, a busca `prefix(X, [1, 2, 3])` deve retornar todos os prefixos da lista `[1, 2, 3]`.

15. Nos próximos exercícios, de 15 até 20, vamos implementar conjuntos como listas. Cada elemento de um conjunto aparece somente uma vez em sua lista, mas as listas não estão necessariamente ordenadas. Você pode assumir que as listas de entrada não possuem elementos duplicados, mas você precisa garantir que as listas de saída não possuem nenhum elemento duplicado. Uma coisa: **não vale usar funções predefinidas de manipulação de listas**, tipo `member`, por exemplo.

    Vamos começar definindo um predicado `isMember`, tal que `isMember(X, Y)` seja verdade se o elemento `X` for estiver presente no conjunto `Y`.

16. Defina um predicado `isUnion`, tal que `isUnion(X, Y, Z)` seja verdade se `Z` for a união de `X` e `Y`. Seu predicado não precisa funcionar direito quando `X` ou `Y` não estão instanciados.

17. Defina um predicado `isIntersection`, tal que `isIntersection(X, Y, Z)` seja verdade se `Z` for for a interseção de `X` e `Y`. Como no caso anterior, seu predicado não precisa funcionar direito quando `X` ou `Y` não estão instanciados.

18. Defina um predicado `isEqual`, tal que `isEqual(X, Y)` seja verdade se `X` e `Y` forem conjuntos iguais. Dois conjuntos são iguais se eles possuem os mesmos elementos, independente da ordem. Novamente, seu predicado não precisa funcionar direito quando `X` ou `Y` não estão instanciados.

19. Defina um predicado `powerSet`, tal que `poserSet(X, Y)` seja verdade se `Y` for o conjunto potência de `X`. O conjunto potência de um dado conjunto s é o conjunto de todos os subconjuntos de s. Por exemplo, se $s = \{1, 2, 3\}$, então o conjunto potência p é:

$$\{x | x \subseteq s\} = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

No caso deste exercício, se `X` é uma lista, representando o conjunto, `Y` será uma lista de listas. Assim, `powerSet([1, 2], Y)` irá produzir a unificação `Y = [[1, 2], [1], [2], []]` (não necessariamente nesta ordem). Seu predicado não precisa funcionar direito quando `X` não estiver instanciado.

20. Defina um predicado `isDifference`, tal que `isDifference(X, Y, Z)` seja verdade se `Z` contiver os elementos de `X` que não aparecem em `Y`. Ao contrário dos exercícios anteriores, seu predicado deverá funcionar independente da ordem em que os elementos de `Z` são dados. Contudo, assim como nos exercícios anteriores, seu predicado não precisa funcionar quando `X` ou `Y` não estiverem instanciados.