Notes

The objective of this project is to practice and assess your understanding of logic programming and Prolog. You will write code to perform the major parts of a cribbage playing program.

Cribbage is a very old card game, dating to early 17th century England. There are 2, 3, and 4 handed versions of the game. The object of the game is to be the first player to reach 121 points. Game play begins with the dealer dealing each player 6 cards (in a 2-player game) or 5 cards (in a 3 or 4 player game). In a 3 player game, the dealer also deals one card to a separate hand called the *crib* or *box*. Each player then chooses 1 or 2 cards to discard, keeping 4 and putting the discarded cards in the *crib* or *box*, which forms a second 4-card hand for the dealer. Next the player preceding the dealer cuts the deck to select an extra card, called the *start* card. If the start card is a Jack, the dealer immediately scores 2 points.

The hand then proceeds to the *play*, wherein the players take turns playing cards from their hands face up in front of them. The play will not be part of this project, so we will not discuss it further.

Following the play comes the *show*, where each player in turn, beginning with the player after the dealer, establishes the value of her hand. For this phase, the start card is usually considered as part of *each* player's hand, so each player establishes the value of a 5 card hand. Points are scored for certain combinations of cards according to the following rules:

- (**15s**) 2 points are scored for each distinct combinations of cards that add to 15. For this purpose, an ace is counted as 1, and a jack, queen or king are counted as 10, and other cards count as their face value. For example, a hand with a 2, 3, 5, 10, and king scores 8 points: 2 points each for 2+3+10, 2+3+king, 5+10, and 5+king.

- (**Pairs**) 2 points are scored for each pair. With 3 of a kind, there are 3 different ways to make a pair, so 3 of a kind scores 6 points. Similarly, 4 of a kind scores 12 points for the 6 possible pairs.
- (**Runs**) 1 point is scored for each card in a *run* of 3 or more consecutive cards (the suits of these cards need not be the same). For example, a 2, 6, 8, and 9, with a 7 as the start card scores 4 points for a 4 card run. It also scores a further 6 points for 15s (6+9, 7+8, and 2+6+7).
- (**Flushes**) 4 points is scored if all the cards in the hand are of the same suit. 1 further point is scored if the start card is also the same suit. Note that no points are scored if 3 of the hand cards, plus the start card, are the same suit.
- (**"One for his nob"**) 1 point is scored if the hand contains the jack of the same suit as the start card.

All of these points are totalled to find the value of a hand. For example (using **A** for ace, **T** for 10, **J** for jack, **Q** for queen, and **K** for king, and ♣, ♦, ♥, and ♠ as clubs, diamonds, hearts, and spades):

Hand	Start card	Points
7♣, Q♥, 2♣, J♣	9♥	0
A♠, 3♥, K♥, 7♥	K♠	2
A♠, 3♥, K♥, 7♥	2♦	5
6♣, 7♣, 8♣, 9♣	8♠	20
7♥, 9♠, 8♣, 7♣	8♥	24
5♥, 5♠, 5♣, J♦	5♦	29

Following the show of each player's hand, the dealer counts the crib as a second hand, claiming any points therein.

Finally, the player following the dealer collects the cards, becoming dealer for the next hand. Play proceeds this way until one player reaches 121 points.

Please see the rules of cribbage for a more complete description of the game. However, what is presented above will be enough to complete the project.

You will write the following predicates:

- `hand_value(Hand, Startcard, Value)`
  `Value` (an integer) is the total cribbage point value of `Hand` when `Startcard` is the start card. `Hand` is represented as a list of 4 card terms, and `Startcard` is a single card term. A card term is a term `card(Rank, Suit)`, where `Rank` is either an integer between 2 and 10, or one of `ace`, `jack`, `queen`, or `king`, and `Suit` is one of `clubs`, `diamonds`, `hearts`, or `spades`. Your code need only work when `Hand` and `Startcard` are ground.
- `select_hand(Cards, Hand, Cribcards)`
  `Cards` is a list of the 5 or 6 cards dealt to a player at the start of a hand. `Hand` is a list of 4 of those cards to be kept to form the players hand, and `Cribcards` is a list of the cards not kept (to be placed in the `crib`). The cards to be kept in the hand should be chosen to maximize the *expected* value of the hand over all possible start cards. Since the start card is not known when the hand is chosen, you cannot be sure to choose the best hand. However the *expected* value of the hand is the average value of the hand over all possible start cards (`Cribcards` and `Hand` cards are `not` possible start cards). This predicate need only work when `Cards` is a ground list of card terms.

Your project will be assessed on the following criteria:

- 50% Correctness of `hand_value`/3 implementation.
- 20% Quality of selections made by the `select_hand`/3 predicate.
- 30% Quality of code and documentation, as determined by markers.

Note that timeouts will be imposed on all tests. You will have at least 1 second per test case for `hand_value`/3 and 4 seconds per test of `select_hand`/3, but executions taking longer than that may be unceremoniously terminated. Several test cases for `hand_value`/3, along with their correct values, are shown earlier in this assignment.

This project is part of your final assessment, so cheating is not acceptable. Any form of material exchange between students, whether written, electronic or any other medium, is considered cheating, and so is the soliciting of help through any medium. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties. If you have questions regarding these rules, please ask the lecturer.

1. It is easier to look for pairs and runs in a hand if it is sorted first, with the start card included. Even 15s are slightly easier to compute when the hand is sorted. However, the start card must not be confused with the cards from the hand when looking for a flush or "one for his nob."
2. This is easier still if the hand is represented as a sorted list of pairs of rank and number of occurrences of that rank. Note that pairs, runs, and 15s depend only on ranks, not on suits.
3. Note that Prolog's built-in sort predicates will not sort card terms correctly. You can either change the representation of cards and use the built-in sort predicate or write your own sort. The easiest way is probably to convert `ace` to 1, `jack` to 11, and so on, and then sort. The built-in predicate `msort`/2 will sort without removing duplicates.
4. Feel free to use your `hand_value`/3 code, or parts of it, in implementing `select_hand`/3.
5. To choose the cards to make up your hand, you can consider the 15 4-card hands you can select from 6 cards, determine their `hand_value`s, and choose the best of these.
6. You can improve on this by considering what the start card may bring to your hand. This will on average do a better job of choosing the best hand. For this, you should consider, for each of the 15 possible hands, the 46 possible start cards and the value it would give your hand. The average of these 46 possibilities would be the number of points you would expect on average to get from that choice of hand. Then simply select the hand with the greatest expected value.