Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2022-2023

Programming Exam 6

Due date: 9 June 2022, Friday, 23.59

1 Problem Definition

After a country-wide decision, it is decided that city councils should be merged in pairs for all cities in the country of Parklandia. Two competing cities, Pawnee and Eagleton are selected as a pair by the Parklandia government, and the city councils of the two cities now need to hold a new election to create a new, merged council.

1.1 Knowledge Base Predicates

All knowledge will be provided in the knowledge base with the following predicates:

```
party(PoliticalParty, Initial).
```

The `party` predicate is used to define political parties that enter the election, where `PoliticalParty` is the name of the party and `Initial` is the one character long initial for the party (used in PART C).

```
candidate(Name, PoliticalParty, City, Row).
```

The `candidate` predicate is used to define candidates and their attributes. Its format is given above, where `Name` is the name of the candidate, `PoliticalParty` is the political party the candidate is associated with, `City` is the name of the city the candidate is from, and `Row` is the row of the candidate in the candidates' list in that city.

```
elected(City, PoliticalParty, ElectedRepresentativeCount).
```

The `elected` predicate is used to define the election results for each city and political party. Its format is given above, where `City` is the name of the city, `PoliticalParty` is the name of the political party, and `ElectedRepresentativeCount` is the count of representatives that are elected from the given City that is associated with the given PoliticalParty.

```
to_elect(RepresentativeCount).
```

The `to_elect` predicate is used to define the count of total representatives to be elected. In other words, `RepresentativeCount` is the size of the new council.

The example knowledge base given to you in kb.pl file can be seen below:

```
party(peoples_parks_party, p).
party(peoples_wealth_party, w).
party(dentists_wealth_party, d).

candidate(leslie_knope, peoples_parks_party, pawnee, 1).
candidate(ben_wyatt, peoples_parks_party, pawnee, 2).
candidate(april_ludgate, peoples_parks_party, pawnee, 3).
candidate(tom_haverford, peoples_parks_party, pawnee, 4).
candidate(jerry_gerwich, peoples_parks_party, pawnee, 5).
candidate(jeremy_jamm, dentists_wealth_party, pawnee, 1).
candidate(joe_fantringham, peoples_wealth_party, pawnee, 1).
candidate(bill_dexhart, peoples_wealth_party, pawnee, 2).
candidate(craig_middlebrooks, peoples_parks_party, eagleton, 1).
candidate(douglass_howser, peoples_parks_party, eagleton, 2).
candidate(ingrid_de_forest, peoples_wealth_party, eagleton, 1).
candidate(lindsay_carlisle_shay, peoples_wealth_party, eagleton, 2).
candidate(george_gernway, peoples_wealth_party, eagleton, 3).
candidate(joan_callamezzo, peoples_wealth_party, eagleton, 4).

elected(pawnee, peoples_parks_party, 4).
elected(eagleton, peoples_wealth_party, 4).
elected(eagleton, peoples_parks_party, 2).

to_elect(10).
```

2 Specifications

In this exam, you are expected to implement different predicates with varying difficulties. All predicates will be queried with different arguments given as variables, but not all arguments will be queried for all predicates. Querying details for each predicate can be seen in the example queries for that predicate.

Assignment is divided into 3 parts. Each part has specific requirements that you should follow.

2.1 PART A - 40 points

This part aims to challenge your understanding of basic concepts of Prolog, such as "not" or "!". You are expected to implement everything by yourselves, in other words, you are not allowed to use any built-in predicates in this part.

2.1.1 `is_vote_wasted/2` - 5 points

Two argument predicate where `City` is the name of the city, `PoliticalParty` is the name of the political party. A vote is wasted if there are no elected candidates from the given City that are candidates of the given political party. Predicate has the form:

```
is_vote_wasted(City, PoliticalParty).
```

Example queries:

```
?- is_vote_wasted(pawnee, dentists_wealth_party).
true.

?- is_vote_wasted(pawnee, peoples_wealth_party).
true.

?- is_vote_wasted(pawnee, peoples_parks_party).
false.
```

- This predicate **will not be queried with its arguments as variables**, so you do not need to worry about the behavior of such queries.

2.1.2 `is_candidate_elected/2` - 15 points

Two argument predicate where `Name` is the name of the candidate, `PoliticalParty` is the name of the political party. A candidate is elected if their Row in the election is smaller than or equal to the elected candidate count from their PoliticalParty in their City. Predicate has the form:

```
is_candidate_elected(Name, PoliticalParty).
```

Example queries:

```
?- is_candidate_elected(leslie_knope, peoples_parks_party).
true.

?- is_candidate_elected(jeremy_jamm, dentists_wealth_party).
false.

?- is_candidate_elected(jerry_gerwich, peoples_parks_party).
false.
```

- This predicate **will not be queried with its arguments as variables**, so you do not need to worry about the behavior of such queries.

2.1.3 `candidate_count_from_city/3` - 20 points

Three argument predicate where `ListOfCandidates` is the list of candidate names, `GivenCity` is the name of the city to check if the candidates are from, and `Count` is the count of candidates in the ListOfCandidates that are from the GivenCity. Predicate has the form:

```
candidate_count_from_city(ListOfCandidates, GivenCity, Count).
```

Example queries:

```
?- candidate_count_from_city([jeremy_jamm, craig_middlebrooks, ben_wyatt, ↩
    ingrid_de_forest], pawnee, Count).
Count = 2.
```

- This predicate **will only be queried with `Count` argument as a variable**, so you do not need to worry about the behavior of other possible queries.

2.2 PART B - 40 points

This part aims to familiarize you with some common built-in predicates in Prolog. You do not necessarily need to use built-in predicates in your implementations, but using them is advised. Some built-in predicates that might come in handy for this part are as follows:

- `findall/3` https://www.swi-prolog.org/pldoc/man?predicate=findall/3
- `length/2` https://www.swi-prolog.org/pldoc/man?predicate=length/2
- `format/2` https://www.swi-prolog.org/pldoc/doc_for?object=format/2
  (For example, `"~2f"` will result in the query being answered in .2f precision.)

2.2.1 `all_parties/1` - 4 points

One argument predicate where `ListOfPoliticalParties` is the list of all political parties that enter the election. Predicate has the form:

```
all_parties(ListOfPoliticalParties).
```

Example queries:

```
?- all_parties(ListOfPoliticalParties).
ListOfPoliticalParties = [peoples_parks_party, peoples_wealth_party, ↩
    dentists_wealth_party].

?- all_parties([peoples_parks_party]).
false.
```

2.2.2 `all_candidates_from_party/2` - 3 points

Two argument predicate where `PoliticalParty` is the name of the PoliticalParty, and `ListOfCandidates` is the list of all candidates that enter the election from the given PoliticalParty. Predicate has the form:

```
all_candidates_from_party(PoliticalParty, ListOfCandidates).
```

Example queries:

```
?- all_candidates_from_party(peoples_parks_party, ListOfCandidates).
ListOfCandidates = [leslie_knope, ben_wyatt, april_ludgate, tom_haverford, ↩
    jerry_gerwich, craig_middlebrooks, douglass_howser].
```

- This predicate **will only be queried with `ListOfCandidates` argument as a variable**, so you do not need to worry about the behavior of other possible queries.

2.2.3 `all_elected_from_party/2` - 3 points

Two argument predicate where `PoliticalParty` is the name of the PoliticalParty, and `ListOfCandidates` is the list of all candidates that are elected from the given PoliticalParty. Predicate has the form:

```
all_elected_from_party(PoliticalParty, ListOfCandidates).
```

Example queries:

```
?- all_elected_from_party(peoples_wealth_party, ListOfCandidates).
ListOfCandidates = [ingrid_de_forest, lindsay_carlisle_shay, george_gernway, ↩
    joan_callamezzo].
```

- This predicate **will only be queried with** `ListOfCandidates` **argument as a variable**, so you do not need to worry about the behavior of other possible queries.

2.2.4 `election_rate/2` - 15 points

Two argument predicate where `PoliticalParty` is the name of the PoliticalParty, and `Percentage` is the ratio of elected candidates from a political party to all candidates from that political party. Predicate has the form:

```
election_rate(PoliticalParty, Percentage).
```

Example queries:

```
?- election_rate(peoples_wealth_party, Percentage).
0.67
Percentage = 0.6666666666666666.

?- election_rate(peoples_parks_party, Percentage).
0.86
Percentage = 0.8571428571428571.
```

- This predicate **will only be queried with** `Percentage` **argument as a variable**, so you do not need to worry about the behavior of other possible queries.

2.2.5 `council_percentage/2` - 15 points

Two argument predicate where `PoliticalParty` is the name of the PoliticalParty, and `Percentage` is the ratio of elected candidates from a political party to count of all representatives that are elected. Predicate has the form:

```
council_percentage(PoliticalParty, Percentage).
```

Example queries:

```
?- council_percentage(peoples_parks_party, Percentage).
0.60
Percentage = 0.6.

?- council_percentage(peoples_wealth_party, Percentage).
0.40
Percentage = 0.4.

?- council_percentage(dentists_wealth_party, Percentage).
0.00
Percentage = 0.
```

- This predicate **will only be queried with** `Percentage` **argument as a variable**, so you do not need to worry about the behavior of other possible queries.

2.3 PART C - 20 points

In this part, you are expected to define helper predicates to make it easy for you to implement the expected predicate. You may also use built-in predicates.
For this part, your goal is to propose alternative sitting plans for the candidate debate that will be hosted. You will be given a string that is composed of the initials of the parties. Given that string, you should propose alternative guest and sitting placements for the debate.

2.3.1 `alternative_debate_setups/2` - 20 points

Two argument predicate where `DescriptionString` is the string composed of the initials of the political parties given in the knowledge base predicate `PoliticalParty, Initial`, and `OrderedListOfCandidates` is the placement list of candidates to be invited to the debate. Predicate has the form:

```
alternative_debate_setups(DescriptionString, OrderedListOfCandidates)
```

Example queries:

```
?- alternative_debate_setups(pdp, OrderedListOfCandidates).
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, craig_middlebrooks] ;
OrderedListOfCandidates = [leslie_knope, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, leslie_knope] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, craig_middlebrooks] ;
OrderedListOfCandidates = [ben_wyatt, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, leslie_knope] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, craig_middlebrooks] ;
OrderedListOfCandidates = [april_ludgate, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, leslie_knope] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, craig_middlebrooks] ;
OrderedListOfCandidates = [tom_haverford, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, leslie_knope] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, craig_middlebrooks] ;
OrderedListOfCandidates = [jerry_gerwich, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, leslie_knope] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [craig_middlebrooks, jeremy_jamm, douglass_howser] ;
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, leslie_knope] ;
```

```
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, ben_wyatt] ;
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, april_ludgate] ;
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, tom_haverford] ;
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, jerry_gerwich] ;
OrderedListOfCandidates = [douglass_howser, jeremy_jamm, craig_middlebrooks] ;
false.
```

- This predicate **will only be queried with `OrderedListOfCandidates` argument as a variable**, so you do not need to worry about the behavior of other possible queries.
- Order of the answers is not important, however, your program should be able to output all the answers when the query is continued with ";".
- Since the candidate names are unique and one person can only be in one place at a time, the outputted list should not have repetitions.

3 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "PE6" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.
- **Programming Language:** You must code your program in Prolog. Your submission will be run with swipl on ODTUCLASS. You are expected to make sure your code runs successfully with swipl on ODTUCLASS.
- **Modules:** You are not allowed to import any modules. However, you are allowed to use the built-in predicates present in Prolog or define your own predicates.
- **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.
- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases.

  Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.