CZ3005 Assignment 3 Report

Zhang Yuhan

U1823060F

SSP2

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Overview

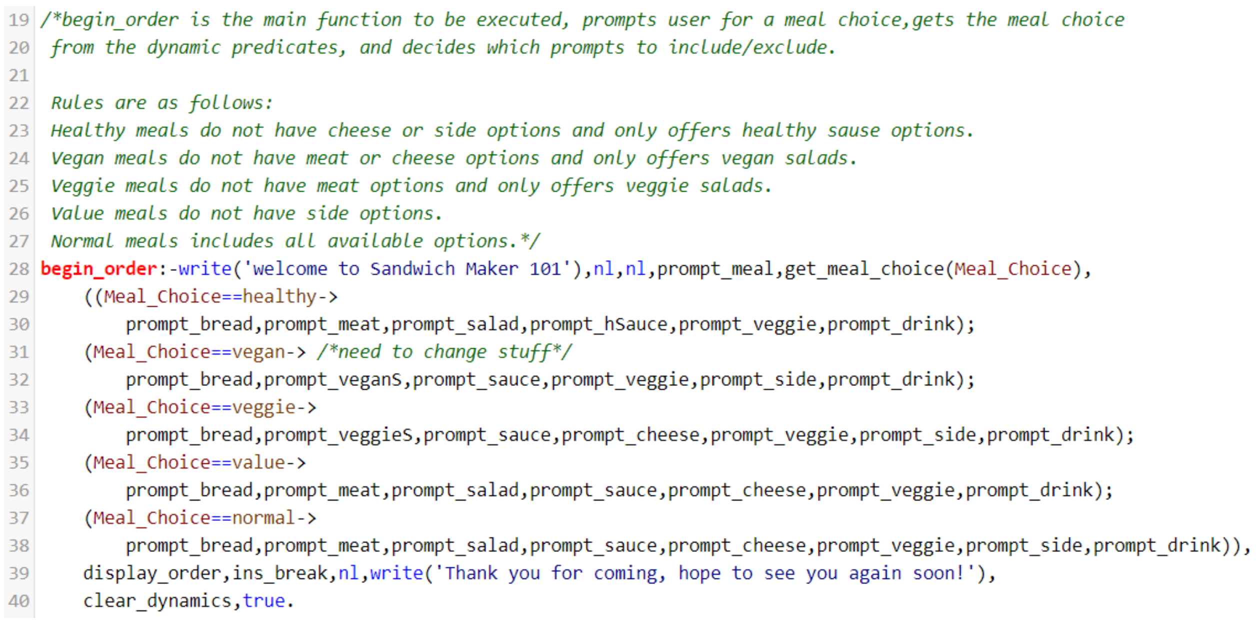
The aim of this assignment is to write a prolog script to mimic the procedures of ordering a sandwich from Subway. The system will provide choices to the user in order to create a customized list from the user's choices. The guidelines of the assignment stated that:

- The script should offer different meal options, sandwich options, meat options, salad options, sauce options, top-up options, sides options etc.
- The options should be intelligently selected based on previous choices.

Implementations of Code

Structure of code:

1. Create predicates for all options in different categories of choices.
2. Create dynamic predicates for storing user choices.
3. Run 'begin_order.' and display a welcoming message and prompt user for a meal type.
4. Depending on the chosen meal type, display the options different categories of choices and prompt for user input.
5. After order is finished, display all user's choices and an ending message.
6. Flush out all elements in dynamic predicates so future runs of the code will not be affected (in case the program is not restarted).

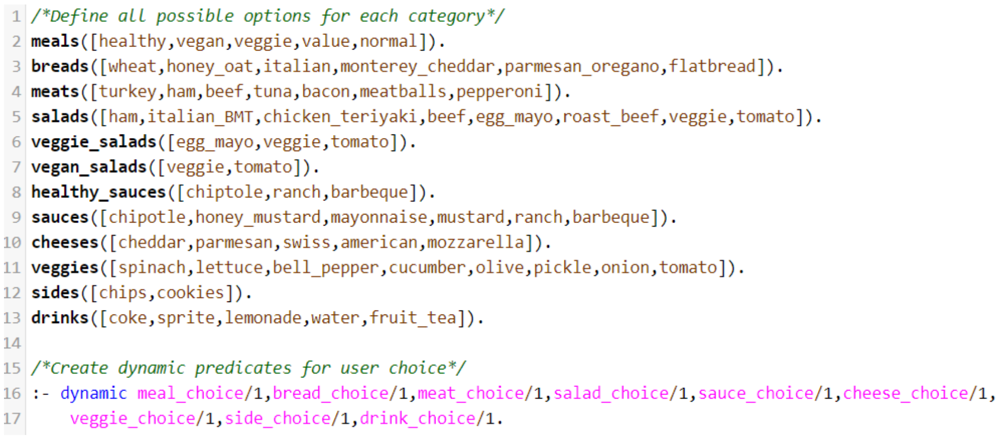


```
/*begin_order is the main function to be executed, prompts user for a meal choice,gets the meal choice
 from the dynamic predicates, and decides which prompts to include/exclude.

 Rules are as follows:
 Healthy meals do not have cheese or side options and only offers healthy sause options.
 Vegan meals do not have meat or cheese options and only offers vegan salads.
 Veggie meals do not have meat options and only offers veggie salads.
 Value meals do not have side options.
 Normal meals includes all available options.*/
begin_order:-write('welcome to Sandwich Maker 101'),nl,nl,prompt_meal,get_meal_choice(Meal_Choice),
    ((Meal_Choice==healthy->
        prompt_bread,prompt_meat,prompt_salad,prompt_hSauce,prompt_veggie,prompt_drink);
    (Meal_Choice==vegan-> /*need to change stuff*/
        prompt_bread,prompt_veganS,prompt_sauce,prompt_veggie,prompt_side,prompt_drink);
    (Meal_Choice==veggie->
        prompt_bread,prompt_veggieS,prompt_sauce,prompt_cheese,prompt_veggie,prompt_side,prompt_drink);
    (Meal_Choice==value->
        prompt_bread,prompt_meat,prompt_salad,prompt_sauce,prompt_cheese,prompt_veggie,prompt_drink);
    (Meal_Choice==normal->
        prompt_bread,prompt_meat,prompt_salad,prompt_sauce,prompt_cheese,prompt_veggie,prompt_side,prompt_drink)),
    display_order,ins_break,nl,write('Thank you for coming, hope to see you again soon!'),
    clear_dynamics,true.
```

Figure 1: General flow of the program.

1, 2.

The following shows how predicates of options in different categories and dynamic predicates are instantiated.

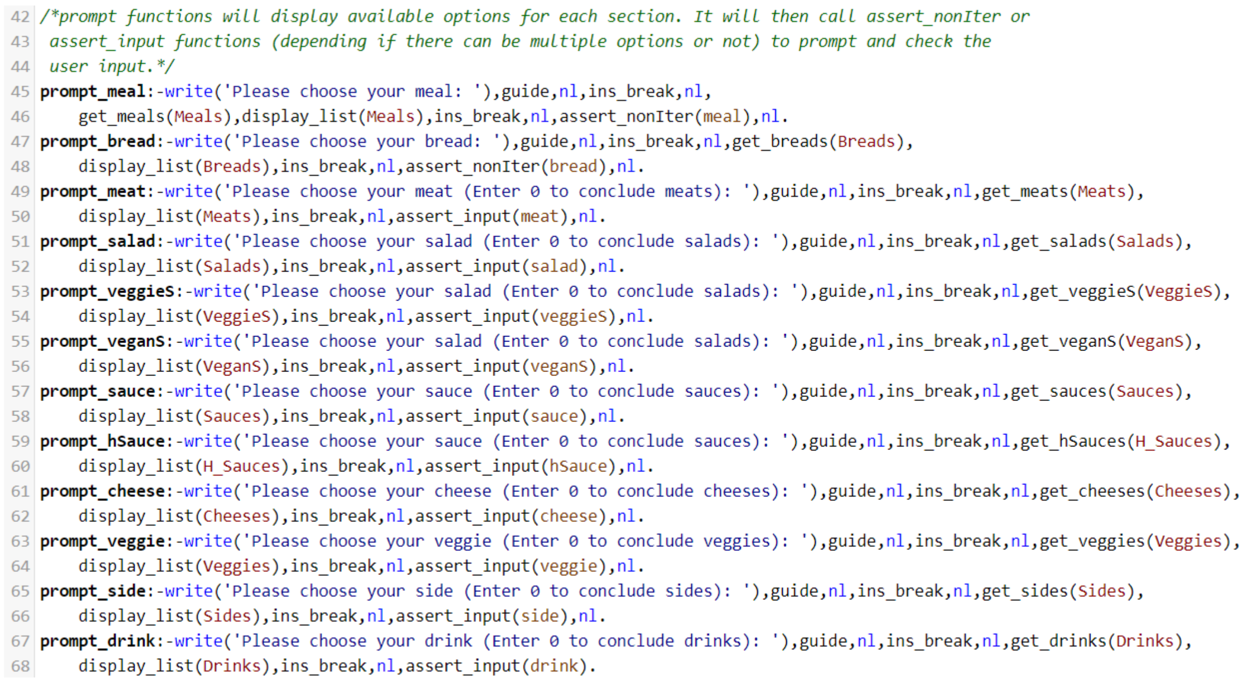


```
/*Define all possible options for each category*/
meals([healthy,vegan,veggie,value,normal]).
breads([wheat,honey_oat,italian,monterey_cheddar,parmesan_oregano,flatbread]).
meats([turkey,ham,beef,tuna,bacon,meatballs,pepperoni]).
salads([ham,italian_BMT,chicken_teriyaki,beef,egg_mayo,roast_beef,veggie,tomato]).
veggie_salads([egg_mayo,veggie,tomato]).
vegan_salads([veggie,tomato]).
healthy_sauces([chiptole,ranch,barbeque]).
sauces([chipotle,honey_mustard,mayonnaise,mustard,ranch,barbeque]).
cheeses([cheddar,parmesan,swiss,american,mozzarella]).
veggies([spinach,lettuce,bell_pepper,cucumber,olive,pickle,onion,tomato]).
sides([chips,cookies]).
drinks([coke,sprite,lemonade,water,fruit_tea]).

/*Create dynamic predicates for user choice*/
:- dynamic meal_choice/1,bread_choice/1,meat_choice/1,salad_choice/1,sauce_choice/1,cheese_choice/1,
    veggie_choice/1,side_choice/1,drink_choice/1.
```

Figure 2: Instantiation of predicates

3.

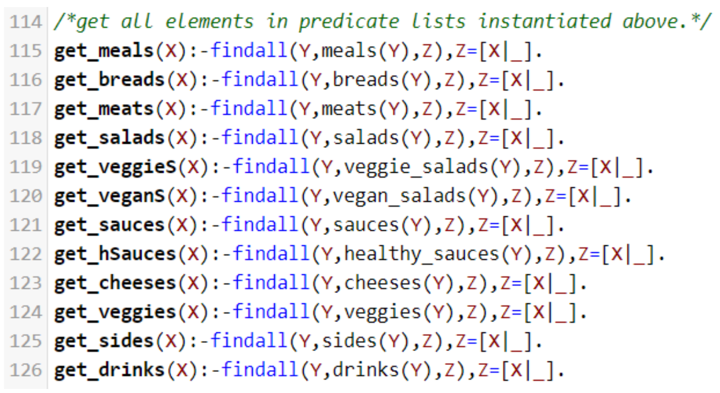
As shown in Figure 1, while executing begin_order, prompt_meal is first called. This function is responsible for getting an input from the user and asserting it into its corresponding dynamic predicate, which in this case would be ‘meal_choice’. This is also true for all other prompt functions in the program as shown below. For simplicity’s sake, I will be using prompt_meal as an example that represents all the prompt functions. The guide and ins_break functions only exist to make the interface cleaner to the user and does not serve any practical purpose.



```
/*prompt functions will display available options for each section. It will then call assert_nonIter or
 assert_input functions (depending if there can be multiple options or not) to prompt and check the
 user input.*/
prompt_meal:-write('Please choose your meal: '),guide,nl,ins_break,nl,
    get_meals(Meals),display_list(Meals),ins_break,nl,assert_nonIter(meal),nl.
prompt_bread:-write('Please choose your bread: '),guide,nl,ins_break,nl,get_breads(Breads),
    display_list(Breads),ins_break,nl,assert_nonIter(bread),nl.
prompt_meat:-write('Please choose your meat (Enter 0 to conclude meats): '),guide,nl,ins_break,nl,get_meats(Meats),
    display_list(Meats),ins_break,nl,assert_input(meat),nl.
prompt_salad:-write('Please choose your salad (Enter 0 to conclude salads): '),guide,nl,ins_break,nl,get_salads(Salads),
    display_list(Salads),ins_break,nl,assert_input(salad),nl.
prompt_veggieS:-write('Please choose your salad (Enter 0 to conclude salads): '),guide,nl,ins_break,nl,get_veggieS(VeggieS),
    display_list(VeggieS),ins_break,nl,assert_input(veggieS),nl.
prompt_veganS:-write('Please choose your salad (Enter 0 to conclude salads): '),guide,nl,ins_break,nl,get_veganS(VeganS),
    display_list(VeganS),ins_break,nl,assert_input(veganS),nl.
prompt_sauce:-write('Please choose your sauce (Enter 0 to conclude sauces): '),guide,nl,ins_break,nl,get_sauces(Sauces),
    display_list(Sauces),ins_break,nl,assert_input(sauce),nl.
prompt_hSauce:-write('Please choose your sauce (Enter 0 to conclude sauces): '),guide,nl,ins_break,nl,get_hSauces(H_Sauces),
    display_list(H_Sauces),ins_break,nl,assert_input(hSauce),nl.
prompt_cheese:-write('Please choose your cheese (Enter 0 to conclude cheeses): '),guide,nl,ins_break,nl,get_cheeses(Cheeses),
    display_list(Cheeses),ins_break,nl,assert_input(cheese),nl.
prompt_veggie:-write('Please choose your veggie (Enter 0 to conclude veggies): '),guide,nl,ins_break,nl,get_veggies(Veggies),
    display_list(Veggies),ins_break,nl,assert_input(veggie),nl.
prompt_side:-write('Please choose your side (Enter 0 to conclude sides): '),guide,nl,ins_break,nl,get_sides(Sides),
    display_list(Sides),ins_break,nl,assert_input(side),nl.
prompt_drink:-write('Please choose your drink (Enter 0 to conclude drinks): '),guide,nl,ins_break,nl,get_drinks(Drinks),
    display_list(Drinks),ins_break,nl,assert_input(drink).
```

Figure 3: Implementation of ‘prompt’ functions

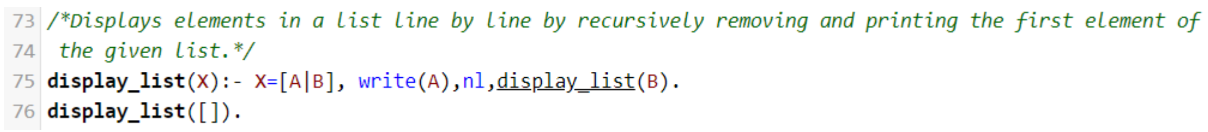
As can be seen from Figure 3, ‘prompt_meal’ first needs to call the function ‘get_meals(X)’ in order to get the list of available options the user can choose from. The predicate findall(Template,Goal,Bag) is used extensively. The logic for the ‘get’ functions is shown below.



```
/*get all elements in predicate lists instantiated above.*/
get_meals(X):-findall(Y,meals(Y),Z),Z=[X|_].
get_breads(X):-findall(Y,breads(Y),Z),Z=[X|_].
get_meats(X):-findall(Y,meats(Y),Z),Z=[X|_].
get_salads(X):-findall(Y,salads(Y),Z),Z=[X|_].
get_veggieS(X):-findall(Y,veggie_salads(Y),Z),Z=[X|_].
get_veganS(X):-findall(Y,vegan_salads(Y),Z),Z=[X|_].
get_sauces(X):-findall(Y,sauces(Y),Z),Z=[X|_].
get_hSauces(X):-findall(Y,healthy_sauces(Y),Z),Z=[X|_].
get_cheeses(X):-findall(Y,cheeses(Y),Z),Z=[X|_].
get_veggies(X):-findall(Y,veggies(Y),Z),Z=[X|_].
get_sides(X):-findall(Y,sides(Y),Z),Z=[X|_].
get_drinks(X):-findall(Y,drinks(Y),Z),Z=[X|_].
```

Figure 4: Implementation of ‘get’ functions

After getting the list of available meal options, ‘display_list(X)’ is executed in order to display the options to the user. The ‘display_list(X)’ is a recursive function that removes and prints the first element of a list, then calls itself with the remaining the list. This continues until there are no more elements left in the list, then ‘display_list([])’. will simply return.



```
/*Displays elements in a list line by line by recursively removing and printing the first element of
 the given list.*/
display_list(X):- X=[A|B], write(A),nl,display_list(B).
display_list([]).
```

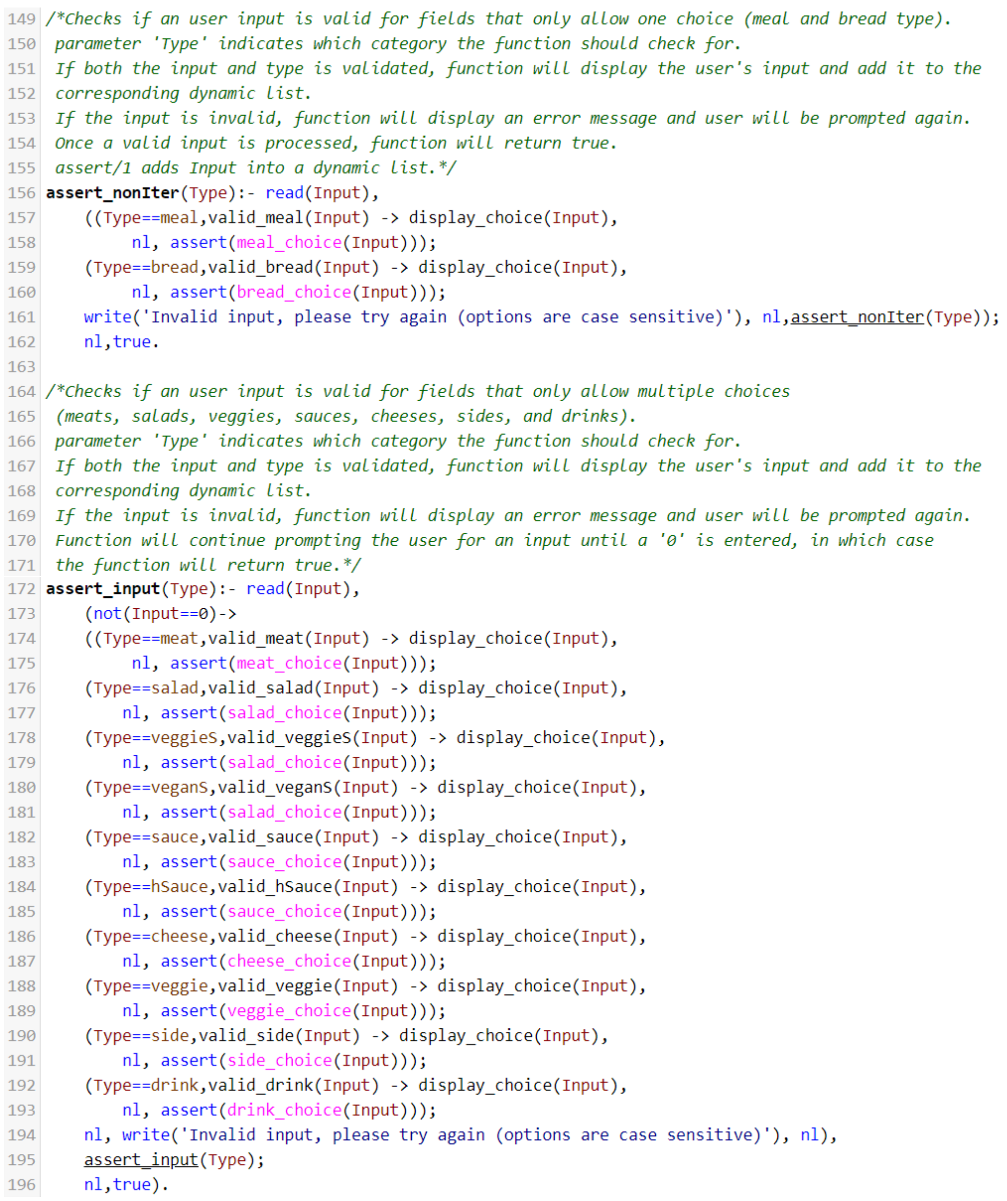
Figure 5: Implementation of ‘display_list(X)’

After all options are displayed, either ‘assert_nonIter(Type)’ or ‘assert_input(Type)’ is called depending on the type/category of the desired input. For example, the input type for ‘prompt_meal’ is ‘meal’ and there should be only one input for meal choice per order, thus ‘assert_nonIter(meal)’ is called. On the other hand, the input type for ‘prompt_meat’ is ‘meat’ and there can be multiple (or zero) selections of meat per order, thus ‘assert_input(meat)’ is called.

Both functions implement similar logic where they first read the user’s input, then checks if the input matches any elements in the predicate list corresponding to the input’s ‘Type’. If there is a match, the user’s choice will be displayed as verification (‘display_choice(Input)’ function) and the input will be asserted into the corresponding dynamic predicate. If the user’s input does not match (invalid input), the function will display an error message and call itself again to repeat the prompt until the user enters a valid input.

The main difference between ‘assert_nonIter(Type)’ and ‘assert_input(Type)’ is that ‘assert_nonIter(Type)’ will stop prompting users for an input after the first valid input is found and return true while ‘assert_input(Type)’ will continue to prompt the user for input as long as the user does not enter ‘0’.

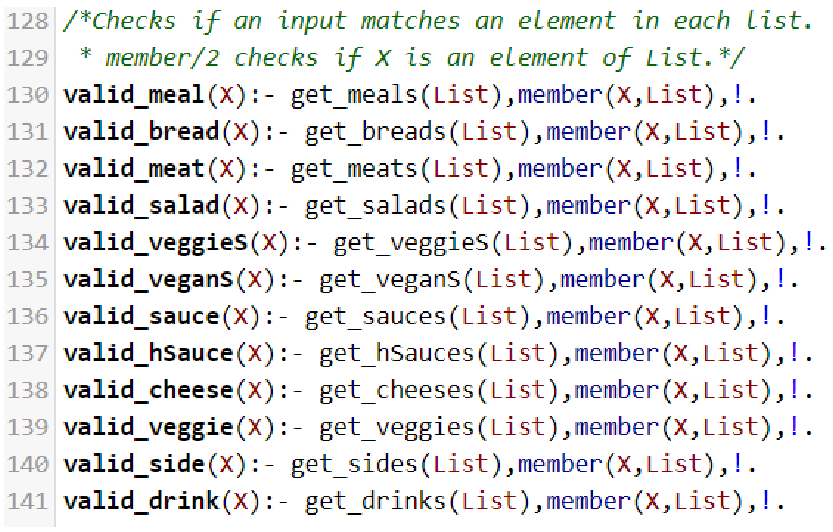
The logic for these functions is shown below.



```prolog
/*Checks if an user input is valid for fields that only allow one choice (meal and bread type).
 parameter 'Type' indicates which category the function should check for.
 If both the input and type is validated, function will display the user's input and add it to the
 corresponding dynamic list.
 If the input is invalid, function will display an error message and user will be prompted again.
 Once a valid input is processed, function will return true.
 assert/1 adds Input into a dynamic list.*/
assert_nonIter(Type):- read(Input),
    ((Type==meal,valid_meal(Input) -> display_choice(Input),
         nl, assert(meal_choice(Input)));
    (Type==bread,valid_bread(Input) -> display_choice(Input),
         nl, assert(bread_choice(Input)));
    write('Invalid input, please try again (options are case sensitive)'), nl,assert_nonIter(Type));
    nl,true.

/*Checks if an user input is valid for fields that only allow multiple choices
 (meats, salads, veggies, sauces, cheeses, sides, and drinks).
 parameter 'Type' indicates which category the function should check for.
 If both the input and type is validated, function will display the user's input and add it to the
 corresponding dynamic list.
 If the input is invalid, function will display an error message and user will be prompted again.
 Function will continue prompting the user for an input until a '0' is entered, in which case
 the function will return true.*/
assert_input(Type):- read(Input),
    (not(Input==0)->
    ((Type==meat,valid_meat(Input) -> display_choice(Input),
         nl, assert(meat_choice(Input)));
    (Type==salad,valid_salad(Input) -> display_choice(Input),
        nl, assert(salad_choice(Input)));
    (Type==veggieS,valid_veggieS(Input) -> display_choice(Input),
        nl, assert(salad_choice(Input)));
    (Type==veganS,valid_veganS(Input) -> display_choice(Input),
        nl, assert(salad_choice(Input)));
    (Type==sauce,valid_sauce(Input) -> display_choice(Input),
        nl, assert(sauce_choice(Input)));
    (Type==hSauce,valid_hSauce(Input) -> display_choice(Input),
        nl, assert(sauce_choice(Input)));
    (Type==cheese,valid_cheese(Input) -> display_choice(Input),
        nl, assert(cheese_choice(Input)));
    (Type==veggie,valid_veggie(Input) -> display_choice(Input),
        nl, assert(veggie_choice(Input)));
    (Type==side,valid_side(Input) -> display_choice(Input),
        nl, assert(side_choice(Input)));
    (Type==drink,valid_drink(Input) -> display_choice(Input),
        nl, assert(drink_choice(Input)));
    nl, write('Invalid input, please try again (options are case sensitive)'), nl),
    assert_input(Type);
    nl,true).
```

Figure 6: Implementation of ‘assert_nonIter(Type)’ and ‘assert_input(Type)’

As can be seen in Figure 6, the user input must be validated against the available options before it can be asserted into the corresponding dynamic predicate. This is done by calling the ‘valid_meat(X)’ function where it gets the list of available meals and compares each element again the input to see if it is a member of the list. The predicate member(element, List) is used to achieve this result.



```prolog
/*Checks if an input matches an element in each list.
 * member/2 checks if X is an element of List.*/
valid_meal(X):- get_meals(List),member(X,List),!.
valid_bread(X):- get_breads(List),member(X,List),!.
valid_meat(X):- get_meats(List),member(X,List),!.
valid_salad(X):- get_salads(List),member(X,List),!.
valid_veggieS(X):- get_veggieS(List),member(X,List),!.
valid_veganS(X):- get_veganS(List),member(X,List),!.
valid_sauce(X):- get_sauces(List),member(X,List),!.
valid_hSauce(X):- get_hSauces(List),member(X,List),!.
valid_cheese(X):- get_cheeses(List),member(X,List),!.
valid_veggie(X):- get_veggies(List),member(X,List),!.
valid_side(X):- get_sides(List),member(X,List),!.
valid_drink(X):- get_drinks(List),member(X,List),!.
```

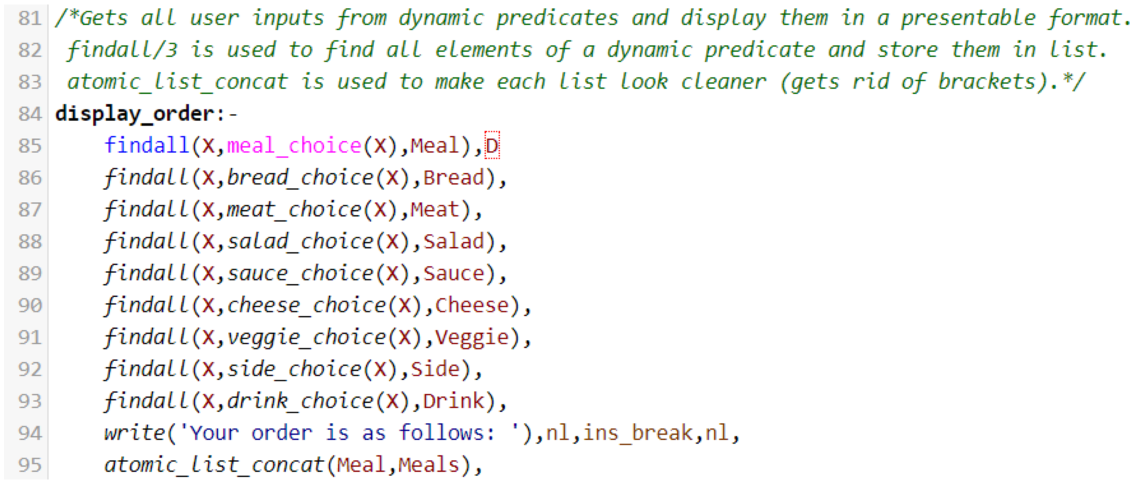
Figure 7: Implementation of ‘valid’ functions

4.

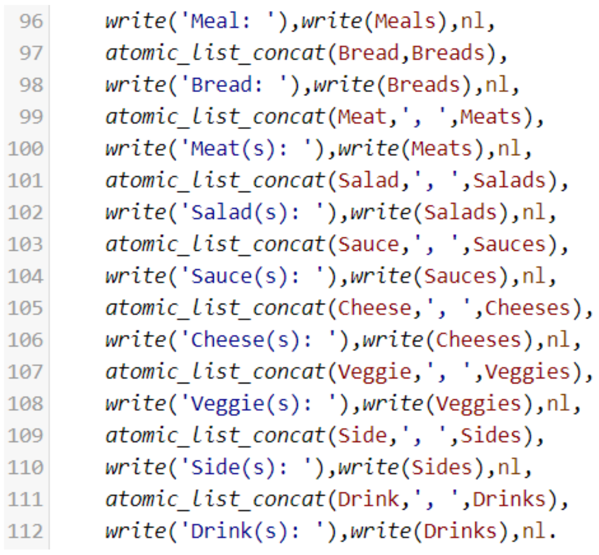
Depending on the user’s input for meal type, the system should include and omit different categories to prompt the user. For example, vegans should not be prompted to enter meat or cheese options and the salad options should only include those that conform with the diet of vegans. The implementation of this can be observed from Figure 1.

5.

After the system has obtain all necessary inputs from the user, it will display the order details to the user. This is done by calling ‘display_order’ predicate where ‘findall(Template,Goal,Bag)’ is used to get the elements in each dynamic predicate. The lists will then be passed through the ‘atomic_list_concat(List,Separator,Atom)’ and printed in the terminal in order for the output to look cleaner. The implementation for ‘display_order’ is shown below.



```prolog
/*Gets all user inputs from dynamic predicates and display them in a presentable format.
 findall/3 is used to find all elements of a dynamic predicate and store them in list.
 atomic_list_concat is used to make each list look cleaner (gets rid of brackets).*/
display_order:-
    findall(X,meal_choice(X),Meal),D
    findall(X,bread_choice(X),Bread),
    findall(X,meat_choice(X),Meat),
    findall(X,salad_choice(X),Salad),
    findall(X,sauce_choice(X),Sauce),
    findall(X,cheese_choice(X),Cheese),
    findall(X,veggie_choice(X),Veggie),
    findall(X,side_choice(X),Side),
    findall(X,drink_choice(X),Drink),
    write('Your order is as follows: '),nl,ins_break,nl,
    atomic_list_concat(Meal,Meals),
```

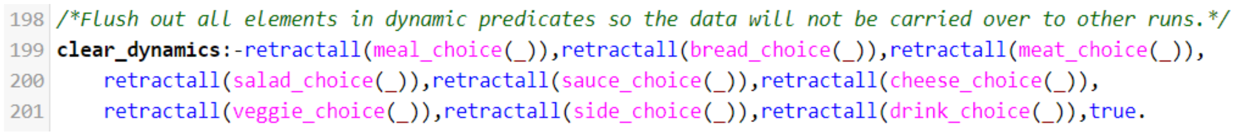


```
    write('Meal: '),write(Meals),nl,
    atomic_list_concat(Bread,Breads),
    write('Bread: '),write(Breads),nl,
    atomic_list_concat(Meat,', ',Meats),
    write('Meat(s): '),write(Meats),nl,
    atomic_list_concat(Salad,', ',Salads),
    write('Salad(s): '),write(Salads),nl,
    atomic_list_concat(Sauce,', ',Sauces),
    write('Sauce(s): '),write(Sauces),nl,
    atomic_list_concat(Cheese,', ',Cheeses),
    write('Cheese(s): '),write(Cheeses),nl,
    atomic_list_concat(Veggie,', ',Veggies),
    write('Veggie(s): '),write(Veggies),nl,
    atomic_list_concat(Side,', ',Sides),
    write('Side(s): '),write(Sides),nl,
    atomic_list_concat(Drink,', ',Drinks),
    write('Drink(s): '),write(Drinks),nl.
```

Figure 8: Implementation of 'display_order'

6.

After the user's order and ending messages are displayed, the dynamic predicates must be flush so that future orders will not be affected. This is shown in the function 'clear_dynamics' in Figure 1. Not flushing the dynamic predicates means that the inputs from the previous order will remain in the dynamic predicates and will not be overwritten by future inputs given that the program has not been restarted. This will result in the wrong lists of user input and may affect other functionalities of the program (such as getting the meal type in 'begin_order'). The implementation of clear_dynamics is shown below.

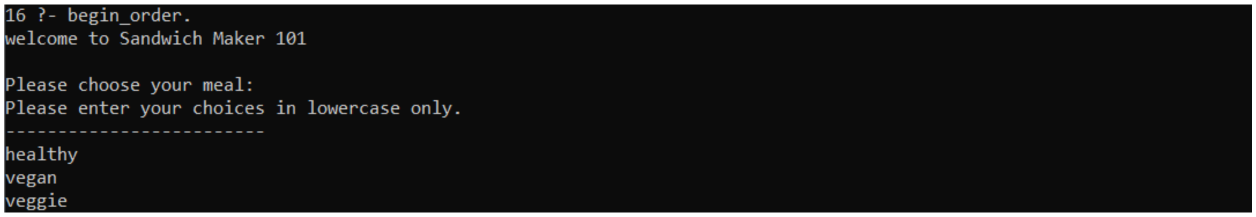


```
/*Flush out all elements in dynamic predicates so the data will not be carried over to other runs.*/
clear_dynamics:-retractall(meal_choice(_)),retractall(bread_choice(_)),retractall(meat_choice(_)),
    retractall(salad_choice(_)),retractall(sauce_choice(_)),retractall(cheese_choice(_)),
    retractall(veggie_choice(_)),retractall(side_choice(_)),retractall(drink_choice(_)),true.
```

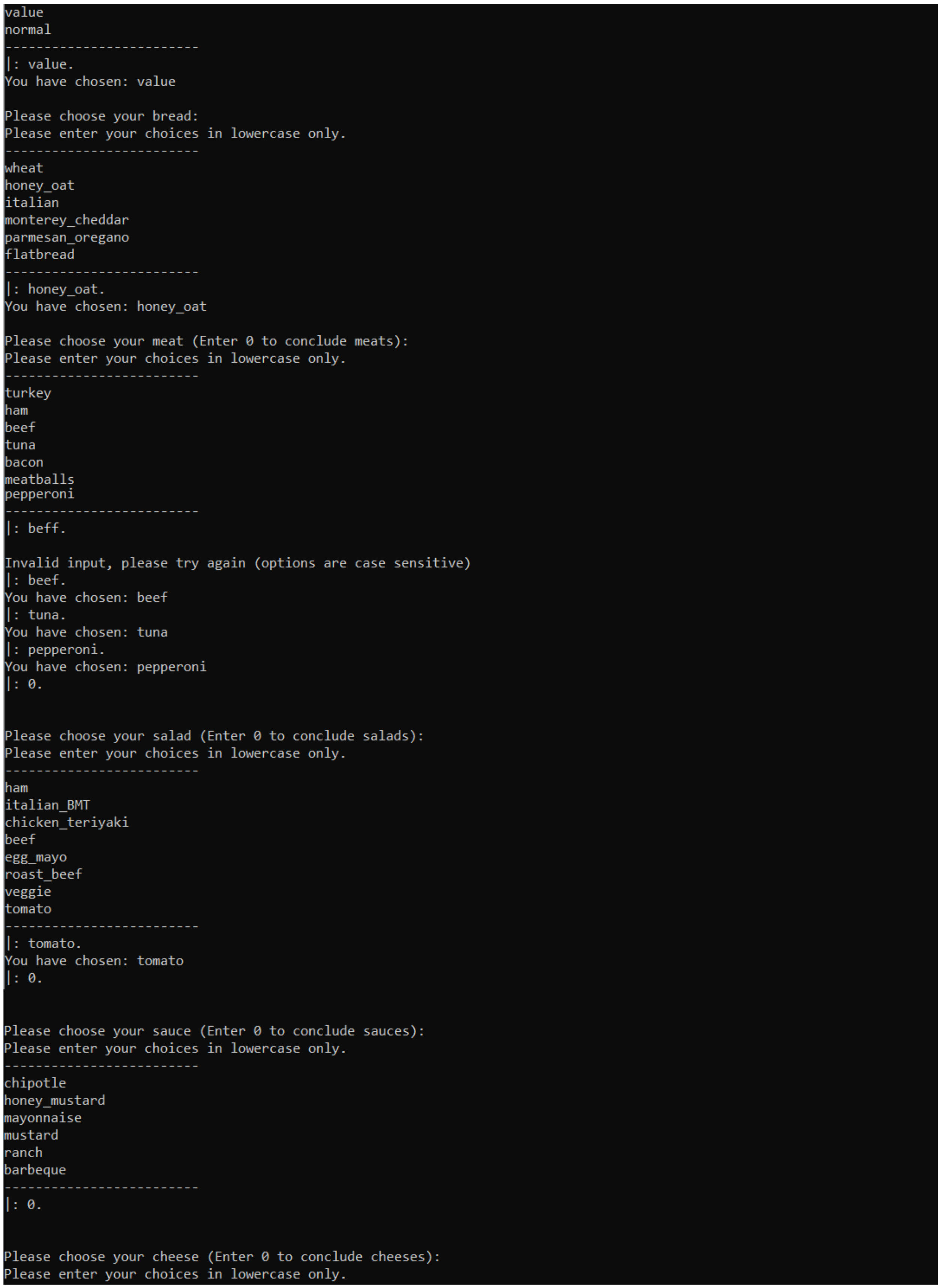
Figure 9: Implementation of 'clear_dynamics'

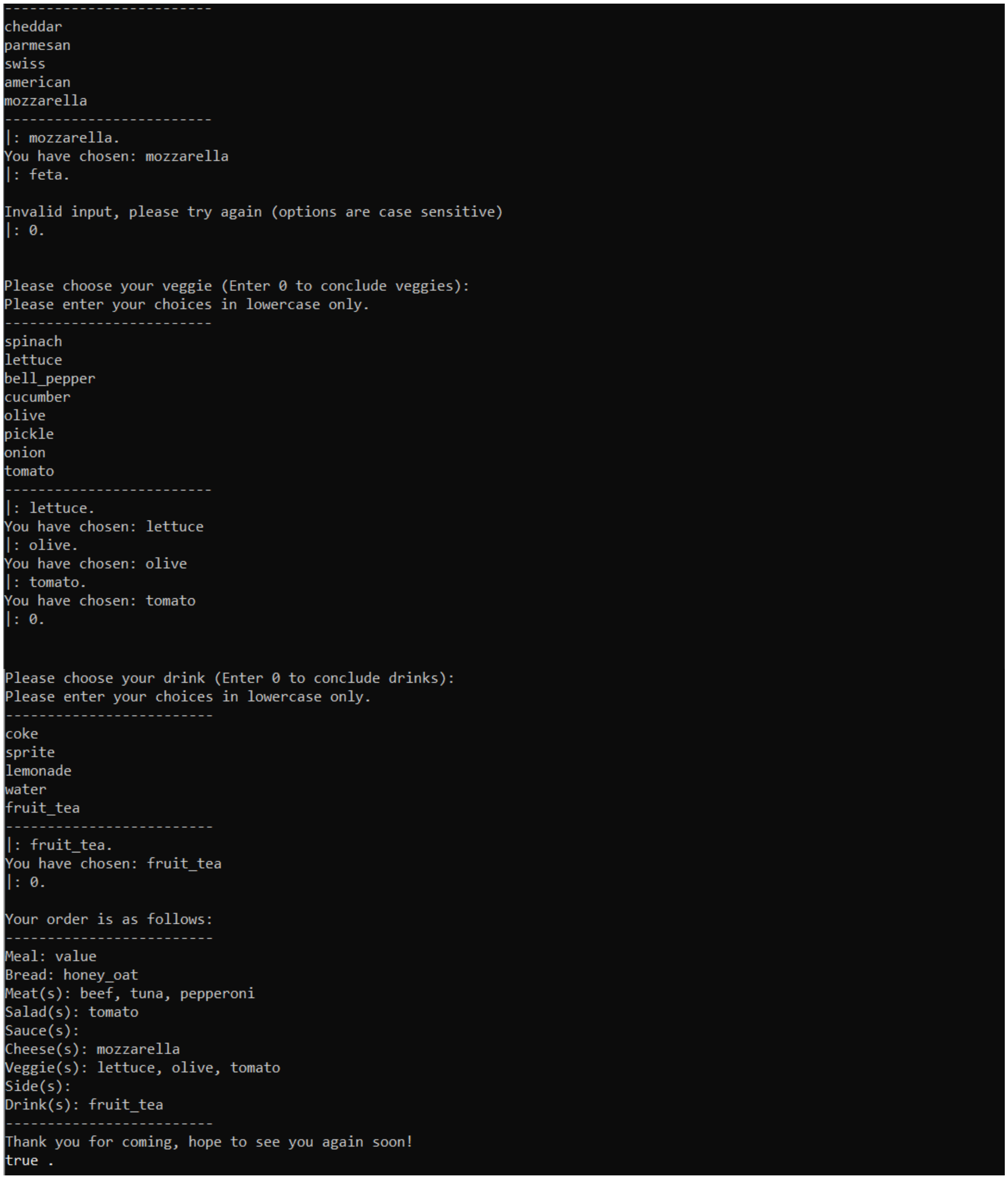
Sample Outputs of Program

Figure 10 below shows the outputs of a run of the 'sandwich_interactor.pl'. In this run, the flow of the program can be clearly observed. Cases of invalid input that requires error handling as well as the system's response to them is also shown.



```
16 ?- begin_order.
Welcome to Sandwich Maker 101

Please choose your meal:
Please enter your choices in lowercase only.
------------------------
healthy
vegan
veggie
```


```
value
normal
-------------------------
|: value.
You have chosen: value

Please choose your bread:
Please enter your choices in lowercase only.
-------------------------
wheat
honey_oat
italian
monterey_cheddar
parmesan_oregano
flatbread
-------------------------
|: honey_oat.
You have chosen: honey_oat

Please choose your meat (Enter 0 to conclude meats):
Please enter your choices in lowercase only.
-------------------------
turkey
ham
beef
tuna
bacon
meatballs
pepperoni
-------------------------
|: beff.

Invalid input, please try again (options are case sensitive)
|: beef.
You have chosen: beef
|: tuna.
You have chosen: tuna
|: pepperoni.
You have chosen: pepperoni
|: 0.


Please choose your salad (Enter 0 to conclude salads):
Please enter your choices in lowercase only.
-------------------------
ham
italian_BMT
chicken_teriyaki
beef
egg_mayo
roast_beef
veggie
tomato
-------------------------
|: tomato.
You have chosen: tomato
|: 0.


Please choose your sauce (Enter 0 to conclude sauces):
Please enter your choices in lowercase only.
-------------------------
chipotle
honey_mustard
mayonnaise
mustard
ranch
barbeque
-------------------------
|: 0.


Please choose your cheese (Enter 0 to conclude cheeses):
Please enter your choices in lowercase only.
```



```
-------------------------
cheddar
parmesan
swiss
american
mozzarella
-------------------------
|: mozzarella.
You have chosen: mozzarella
|: feta.

Invalid input, please try again (options are case sensitive)
|: 0.


Please choose your veggie (Enter 0 to conclude veggies):
Please enter your choices in lowercase only.
-------------------------
spinach
lettuce
bell_pepper
cucumber
olive
pickle
onion
tomato
-------------------------
|: lettuce.
You have chosen: lettuce
|: olive.
You have chosen: olive
|: tomato.
You have chosen: tomato
|: 0.


Please choose your drink (Enter 0 to conclude drinks):
Please enter your choices in lowercase only.
-------------------------
coke
sprite
lemonade
water
fruit_tea
-------------------------
|: fruit_tea.
You have chosen: fruit_tea
|: 0.

Your order is as follows:
-------------------------
Meal: value
Bread: honey_oat
Meat(s): beef, tuna, pepperoni
Salad(s): tomato
Sauce(s):
Cheese(s): mozzarella
Veggie(s): lettuce, olive, tomato
Side(s):
Drink(s): fruit_tea
-------------------------
Thank you for coming, hope to see you again soon!
true .
```

Figure 10: Prolog output from running ‘begin_order.’