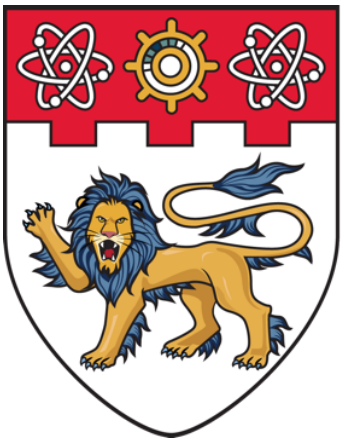

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Talking Box

Assignment 3:

Subway Sandwich Interactor

Lab Report

BCG3
Hans Tananda U1720251K

Implementation Explanation

In this assignment, I used Prolog as the Knowledge Base as well as the interface for the "Subway Sandwich Interactor". The implementation is built under a prolog script named '*subway.pl*' . The program will be started after calling the predicate '*start/0*' .

For the list of breads, meats, vegetables, as well as other options are taken based on the official Subway Singapore Website[1]. I also looked through some more websites to get information regarding list of healthy sauces[2] and vegan sauces[3] . Below is the part of the code that declares the facts.

```
breads([italian_wheat, hearty_italian, honey_oat, parmesan_oregano,
flatbread, multigrain]).
meats([ turkey_breast, ham, chicken_breast, roast_beef, tuna,
turkey_salami, beefsteak, bacon, meatballs, pepperoni]).
cheeses([processed_cheddar, monterey_cheddar, none]).
vegs([cucumbers, green_bell_peppers, lettuce, red_onions, tomatoes,
black_olives, jalapeno, pickles]).
sauces([honey_mustard, yelow_mustard, deli_brown_mustard, sweet_onion,
chipotle_southwest, ranch, bbq, chili_sauce, tomato_sauce, mayonnaise]).
vegan_sauces([chili_sauce, tomato_sauce]).
healthy_sauces([honey_mustard, yelow_mustard, deli_brown_mustard,
sweet_onion]).
sides([chips, cookies, hashbrowns, energy_bar_and_fruit_crisps,
yogurt]).
drinks([fountain_drinks, dasani_mineral_water,
minute_maid_puly_orange_juice, ayataka_japanese_green_tea, coffee,
tea]).
salads([cold_cut_trio, chicken_and_bacon_ranch, chicken_teriyaki,
egg_mayo, chicken_ham, italian_bmt, meatball_marinara_melt, roast_beef,
veggie_party]).
```

[1] "Home | SUBWAY.com - Singapore (English)." https://www.subway.com/en-SG. Accessed 6 Apr. 2019.
[2] "Eating Healthy at Subway | SparkPeople." https://www.sparkpeople.com/resource/sparkdining-eatery.asp?id=6. Accessed 6 Apr. 2019.
[3] "How to Eat Vegan at Subway | PETA." 8 Sep. 2016, https://www.peta.org/living/food/eat-vegan-subway/. Accessed 6 Apr. 2019.

To print the list based on the respective options, I created a predicate *'print_options'* that will print the items in a list one by one. Then, another rule named *'options/1'* is created to take the fact list based on given argument.

```
% print_options is used to print
the items based on the given list.
print_options([]). % empty list
print_options([H]) :- % last item
in list
       write(H),
       write('.').

print_options([H|T]) :-  % List
with items more than one
       write(H),
       write(', '),
       print_options(T), !.% remove
the item then print it one by one

% options is used to choose the
list based on the arguments given
and call the print list command
options(breads):- % Print options
for breads
       breads(L), print_options(L).
% Get the list and then call the
print function with the respective
list

options(meats):- % Functionally the
same as above but for meats
       meats(L), print_options(L).

options(cheeses):- % Functionally
the same as above but for cheeses
       cheeses(L), print_options(L).
```

```
options(vegs):- % Functionally the
same as above but for vegetables
       vegs(L), print_options(L).

options(sauces):- % Functionally
the same as above but for sauces
       sauces(L), print_options(L).

options(vegan_sauces):- %
Functionally the same as above but
for vegan sauces

vegan_sauces(L),print_options(L).

options(healthy_sauces):- %
Functionally the same as above but
for healthy sauces

healthy_sauces(L),print_options(L).

options(sides):- % Functionally the
same as above but for sides
       sides(L), print_options(L).

options(drinks):- % Functionally
the same as above but for drinks
       drinks(L), print_options(L).

options(salads):- % Functionally
the same as above but for salads
       salads(L), print_options(L).
```

Then, to assert the fact, I created a rule named '*selected/2*' that is used to assert the fact dynamically[4] based on the given item X and the specific fact list. '*check_selection/2*' rule is also added to check whether the given item is in the specific fact list. Another predicate *'print_selected'* is used to find all the asserted facts on a specific option and print them.

[4] "Dynamic Switch Case in Prolog - Stack Overflow." 27 May. 2016, https://stackoverflow.com/questions/37463304/dynamic-switch-case-in-prolog. Accessed 6 Apr. 2019.

```
% Check selection is used to check
whether the given input is inside
the list of fact specified
% '!' is used to cut the
backtracking and return right away
when the
check_selection(X, breads):- %
Check X is in list of breads
       breads(L), member(X,L),!. %
For the given list, check whether
element is in a list

check_selection(X, meats):- % Check
X is in list of meats
       meats(L), member(X,L),!.

check_selection(X, cheeses):- %
Check X is in list of cheeses
       cheeses(L), member(X,L),!.

check_selection(X, vegs):- % Check
X is in list of vegetables
       vegs(L), member(X,L),!.

check_selection(X, vegan_sauces):-
% Check X is in list of vegan
sauces
       vegan_sauces(L),
member(X,L),!.

check_selection(X,
healthy_sauces):- % Check X is in
list of healthy sauces
       healthy_sauces(L),
member(X,L),!.

check_selection(X, sauces):- %
Check X is in list of sauces
       sauces(L), member(X,L),!.

check_selection(X, sides):- % Check
X is in list of sides
       sides(L), member(X,L),!.

check_selection(X, drinks):- %
Check X is in list of drinks
       drinks(L), member(X,L),!.

check_selection(X,salads):- % Check
X is in list of salads
       salads(L), member(X,L),!.

% Selected is used to assert the
facts given the item argument(X)
and the specific list of fact
selected(X,breads):- % assert a
bread fact from the given argument
and print the value
       assert(bread(X)),
print_selected(breads).

selected(X,meats):- % assert a meat
fact from the given argument and
print the value
       assert(meat(X)),
print_selected(meats).

selected(X,cheeses):- % assert a
cheese fact from the given argument
and print the value
       assert(cheese(X)),
print_selected(cheeses).

selected(X,vegs):- % assert a
vegetable fact from the given
argument and print the value
       assert(veg(X)),
print_selected(vegetables).

selected(X,sauces):- % assert a
sauce fact from the given argument
and print the value
       assert(sauce(X)),
print_selected(sauces).

selected(X,sides):- % assert a side
fact from the given argument and
print the value

assert(side(X)),print_selected(side
s).

selected(X,drinks):- % assert a
drink fact from the given argument
and print the value
       assert(drink(X)),
print_selected(drinks).

selected(X,salads):- % assert a
```

```
salad fact from the given argument
and print the value
      assert(salad(X)),
print_selected(salads).

% Print selected is used to find
asserted items of a specific list
and then print them.
print_selected(breads):- % print
the asserted breads
      findall(X, bread(X), Breads),
atomic_list_concat(Breads,
',',Bread), % Find the items and
concat it into one string
      write("Bread selected:
"),write(Bread),write("."),nl. %
Print the items

print_selected(meats):- % print the
asserted meats
      findall(X, meat(X), Meats),
atomic_list_concat(Meats,
',',Meat),
      write("Meats selected:
"),write(Meat),write("."),nl.

print_selected(cheeses):- % print
the asserted cheeses
      findall(X, cheese(X),
Cheeses),
atomic_list_concat(Cheeses,
',',Cheese),
      write("Cheese selected:
"),write(Cheese),write("."),nl.

print_selected(vegetables):- %
print the asserted vegetables
      findall(X, veg(X), Vegs),
atomic_list_concat(Vegs, ',',Veg),
      write("Vegetables selected:
"),write(Veg),write("."),nl.

print_selected(sauces):- % print
the asserted sauces
      findall(X, sauce(X), Sauces),
atomic_list_concat(Sauces,
',',Sauce),
      write("Sauces selected:
"),write(Sauce),write("."),nl.

print_selected(sides):- % print the
asserted sides
      findall(X, side(X), Sides),
atomic_list_concat(Sides,
',',Side),
      write("Sides selected:
"),write(Side),write("."),nl.

print_selected(salads):- % print
the asserted salads
      findall(X, salad(X), Salads),
atomic_list_concat(Salads,
',',Salad),
      write("Salads selected:
"),write(Salad),write("."),nl.

print_selected(drinks):- % print
the asserted drinks
      findall(X, drink(X), Drinks),
atomic_list_concat(Drinks,
',',Drink),
      write("Drink selected:
"),write(Drink),write("."),nl.
```

To query the user, I created *'query/1'* predicate that will print the available options and ask the user to pick. If the one that is picked by the user is not in the list, it will loops back again. For some queries that allows the user to choose more than one options (like vegetables and sauces), it will loops back until the user inputs '0'. I also declared some of the predicates as dynamic so that it can be asserted dynamically on runtime. Then, healthy and vegan sauces are validated based on its respective list, however, the one that is asserted is the sauce itself. My rationale for this decision is that the healthy and vegan sauces are the subset of all of the sauces, and less switches will be used by doing this.

```
% Query is used to get the user
inputs for each of the options
query(bread):- % Get the type of
bread selected by the user
	print("Please choose your
bread type:"),
	options(breads),nl,
	read(X),
	check_selection(X,breads) ->
selected(X,breads); % If the input
given is valid, then assert the
fact
	write("Your selection is not
in the list! Please try
again..."),nl,
	query(bread). % The input is
invalid, thus it loops back to
query the user again

query(cheese):- % Get the type of
cheese selected by the user
	print("Please choose your
type of cheese:"),
	options(cheeses),nl,
	read(X),
	check_selection(X,cheeses) ->
selected(X,cheeses);
	write("Your selection is not
in the list! Please try
again..."),nl,
	query(cheese).

query(drink):- % Get the type of
drink selected by the user
	print("Please choose your
drink:"),
	options(drinks),nl,
	read(X),
	check_selection(X,drinks) ->
selected(X,drinks);
	write("Your selection is not
in the list! Please try
again..."),nl,
	query(drink).

query(meat):- % Get the type of
meats selected by the user
	print("Please choose the
meats you want one by one(0 to
end):"),
	options(meats),nl,
	read(X),
	not(X==0) ->
	(check_selection(X,meats) ->
selected(X,meats); % If the input
given is valid, then assert the
fact
		write("Your selection
is not in the list! Please try
again..."),nl), % Input is invalid,
just inform the user
	query(meat); % Loops back as
long as the input is not 0,
regardless whether the input is
valid or invalid
	true. % Ends the loop if the
input is 0

query(veg) :- % Get the type of
vegetables selected by the user
	print("Please choose the
vegetables you want one by one(0 to
end):"),
	options(vegs),nl,
	read(X),
	not(X==0) ->
	( check_selection(X, vegs) ->
selected(X,vegs);
		write("Your selection
```

```
is not in the list! Please try
again..."),nl ),
       query(veg);
       true.

query(healthy_sauce):- % Get the
type of healthy sauces selected by
the user
       print("Please choose the type
of sauces you want one by one(0 to
end):"),
       options(healthy_sauces),nl,
       read(X),
       not(X==0) ->
       (
check_selection(X,healthy_sauces)
-> selected(X,sauces); % The one
that is asserted is sauce for
simplification, since healthy
sauces are a subset of sauces
              write("Your selection
is not in the list! Please try
again..."),nl),
       query(healthy_sauce);
       true.

query(vegan_sauce):- % Get the type
of vegan sauces selected by the
user
       print("Please choose the type
of sauces you want one by one(0 to
end):"),
       options(vegan_sauces),nl,
       read(X),
       not(X==0) ->
       ( check_selection(X,
vegan_sauces) ->
selected(X,sauces); % The one that
is asserted is sauce for
simplification, since vegan sauces
are a subset of sauces
              write("Your selection
is not in the list! Please try
again..."),nl ),
       query(vegan_sauce);
       true.

query(sauce):- % Get the type of
sauces selected by the user
       print("Please choose the type
of sauces you want one by one(0 to
end):"),
       options(sauces),nl,
       read(X),
       not(X==0) ->
       ( check_selection(X, sauces)
-> selected(X,sauces);
              write("Your selection
is not in the list! Please try
again..."),nl ),
       query(sauce);
       true.

query(side):- % Get the type of
sides selected by the user
       print("Please choose the
sides you want one by one(0 to
end):"),
       options(sides),nl,
       read(X),
       not(X==0) ->
       ( check_selection(X, sides)
-> selected(X,sides);
              write("Your selection
is not in the list! Please try
again..."),nl ),
       query(side);
       true.

query(salad):- % Get the type of
salads selected by the user
       print("Please choose the
salads you want one by one(0 to
end):"),
       options(salads),nl,
       read(X),
       not(X==0) ->
       ( check_selection(X, salads)
-> selected(X,salads);
              write("Your selection
is not in the list! Please try
again..."),nl ),
       query(salad);
       True.

% Declare dynamic predicates to
store results
:- dynamic bread/1, meat/1, veg/1,
sauce/1, side/1, drink/1, cheese/1,
salad/1, meal_type/1.
```

To query the user based on what type of meal they wanted, '*meal/1*' predicate is added so that it can call specific queries based on the meal type the user wants. This will be called by '*start/0*', which is the entry point of the main program. *'start'* predicate will query the user of which meal type the user wants, and then call the respective meal rule. It will also loops back if the user input is not in the list of meal type available.

```
% This predicate is used to
intelligently choose the
appropriate queries for the user
meal(normal):- % Normal meal will
show the query of all options
except salad
      query(bread), query(meat),
query(veg), query(cheese),
query(sauce), query(side),
query(drink).
meal(veggie):- % Veggie meal will
remove the meats options from the
normal meal
      query(bread), query(veg),
query(cheese), query(sauce),
query(side), query(drink).
meal(vegan):- % Vegan meal will not
ask cheese and meats options and
query only vegan sauces
      query(bread), query(veg),
query(vegan_sauce), query(side),
query(drink).
meal(healthy):- % Healthy meal will
only query for healty sauces and
not ask for sides
      query(bread), query(meat),
query(veg), query(healthy_sauce),
query(drink).
meal(value) :- % Value meal will
not query on sides and drinks
      query(bread), query(meat),
query(veg), query(cheese),
query(sauce).
meal(salad):- % Salad meal will
only query salad options
      query(salad), query(side),
query(drink).

start:- % Start the program
      preprint,
      write("Choose meal
type:(normal, veggie, vegan,
healthy, value, salad)"),nl,
      read(Type),
      (Type== veggie -> % Given
input is veggie, call the queries
for veggie meal and assert
meal_type to veggie
      write("meal type :"),
write(Type),nl,
      meal(veggie),
assert(meal_type(veggie));
      Type== vegan -> % Given input
is vegan, call the queries for
vegan meal and assert meal_type to
vegan
      write("meal type :"),
write(Type),nl,
      meal(vegan),
assert(meal_type(vegan));
      Type== healthy -> % Given
input is healthy, call the queries
for healthy  meal and assert
meal_type to healthy
      write("meal type :"),
write(Type),nl,
      meal(healthy),
assert(meal_type(healthy));
      Type== value -> % Given input
is value, call the queries for
value meal and assert meal_type to
value
      write("meal type :"),
write(Type),nl,
      meal(value),
assert(meal_type(value));
      Type== normal -> % Given
input is normal, call the queries
for normal meal and assert
meal_type to normal
      write("meal type :"),
write(Type),nl,
      meal(normal),
```

```
assert(meal_type(normal));
      Type== salad -> % Given input
is salad, call the queries for
salad meal and assert meal_type to
salad
      write("meal type :"),
write(Type),nl,
      meal(salad),
assert(meal_type(meal(salad)));
      write("invalid option
selected!"),nl,
      start), % Loops back to query
the user again about the type if it
is not in the list
      display,
      postprint,
      reset.
```

Finally, I created additional predicates to print the result named *'display/0'*, as well as to reset all the asserted facts named *'reset/0'*. For the reset, I used a *'fail/0'* predicate so that it can call other reset functions as well after resetting a certain options. In addition, I created *'preprint/0'* and *'postprint/0'* to display welcome and ending messages respectively.

```
display:- % Display all the
selected facts
      meal_type(Meal), % Find the
meal type and print it
      write("Meal type selected:
"), write(Meal), nl,
      write("Choices
selected:"),nl,
      print_selected(breads), %
Calling all the respective print
function above
      print_selected(meats),
      print_selected(cheeses),
      print_selected(vegetables),
      print_selected(sauces),
      print_selected(sides),
      print_selected(salads),
      print_selected(drinks).

preprint:- % Welcome message

write("============================
==============================="),n
l,
      write("====================
WELCOME TO SUBWAY
===================="),nl,

write("============================
==============================="),n
l.
```

```
postprint:- % Ending message

write("============================
==============================="),n
l,

write("========================
THANK YOU
========================"),nl,
      write("============== PLEASE
COME AND USE ME AGAIN :D
============"),nl,

write("============================
==============================="),n
l.

% Reset is used to remove the
assertion of the dynamic facts
reset:- retractall(bread(_)),fail.
% fail is used to force the
travelsal continue after
successfully deleting specific
facts
reset:- retractall(meat(_)),fail.
reset:- retractall(veg(_)),fail.
reset:- retractall(sauce(_)),fail.
reset:- retractall(salad(_)),fail.
reset:- retractall(cheese(_)),fail.
reset:- retractall(side(_)),fail.
reset:- retractall(drink(_)),fail.
reset:- retractall(meal_type(_)).
```

Snippet of Program Execution

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- start.
============================================================
=================== WELCOME TO SUBWAY ====================
============================================================
Choose meal type:(normal, veggie, vegan, healthy, value, salad)
|: normal.
meal type :normal
"Please choose your bread type:"italian_wheat, hearty_italian, honey_oat, parmesan_oregano, flatbread,
multigrain.
|: honey_oat.
Bread selected: honey_oat.
"Please choose the meats you want one by one(0 to end):"turkey_breast, ham, chicken_breast, roast_beef,
 tuna, turkey_salami, beefsteak, bacon, meatballs, pepperoni.
|: bacon.
Meats selected: bacon.
"Please choose the meats you want one by one(0 to end):"turkey_breast, ham, chicken_breast, roast_beef,
 tuna, turkey_salami, beefsteak, bacon, meatballs, pepperoni.
|: ham.
Meats selected: bacon,ham.
"Please choose the meats you want one by one(0 to end):"turkey_breast, ham, chicken_breast, roast_beef,
 tuna, turkey_salami, beefsteak, bacon, meatballs, pepperoni.
|: 0.
"Please choose the vegetables you want one by one(0 to end):"cucumbers, green_bell_peppers, lettuce, re
d_onions, tomatoes, black_olives, jalapeno, pickles.
|: lettuce.
Vegetables selected: lettuce.
"Please choose the vegetables you want one by one(0 to end):"cucumbers, green_bell_peppers, lettuce, re
d_onions, tomatoes, black_olives, jalapeno, pickles.
|: chili.
Your selection is not in the list! Please try again...
"Please choose the vegetables you want one by one(0 to end):"cucumbers, green_bell_peppers, lettuce, re
d_onions, tomatoes, black_olives, jalapeno, pickles.
|: jalapeno.
Vegetables selected: lettuce,jalapeno.
"Please choose the vegetables you want one by one(0 to end):"cucumbers, green_bell_peppers, lettuce, re
d_onions, tomatoes, black_olives, jalapeno, pickles.
|: 0.
"Please choose your type of cheese:"processed_cheddar, monterey_cheddar, none.
|: monterey_cheddar.
Cheese selected: monterey_cheddar.
"Please choose the type of sauces you want one by one(0 to end):"honey_mustard, yelow_mustard, deli_bro
wn_mustard, sweet_onion, chipotle_southwest, ranch, bbq, chili_sauce, tomato_sauce, mayonnaise.
|: bbq.
Sauces selected: bbq.
```

```
"Please choose the type of sauces you want one by one(0 to end):"honey_mustard, yelow_mustard, deli_bro
wn_mustard, sweet_onion, chipotle_southwest, ranch, bbq, chili_sauce, tomato_sauce, mayonnaise.
|: ranch.
Sauces selected: bbq,ranch.
"Please choose the type of sauces you want one by one(0 to end):"honey_mustard, yelow_mustard, deli_bro
wn_mustard, sweet_onion, chipotle_southwest, ranch, bbq, chili_sauce, tomato_sauce, mayonnaise.
|: 0.
"Please choose the sides you want one by one(0 to end):"chips, cookies, hashbrowns, energy_bar_and_frui
t_crisps, yogurt.
|: chips.
Sides selected: chips.
"Please choose the sides you want one by one(0 to end):"chips, cookies, hashbrowns, energy_bar_and_frui
t_crisps, yogurt.
|: 0.
"Please choose your drink:"fountain_drinks, dasani_mineral_water, minute_maid_puly_orange_juice, ayatak
a_japanese_green_tea, coffee, tea.
|: dasani_mineral_water.
Drink selected: dasani_mineral_water.
============================================================
Meal type selected: normal
Choices selected:
Bread selected: honey_oat.
Meats selected: bacon,ham.
Cheese selected: monterey_cheddar.
Vegetables selected: lettuce,jalapeno.
Sauces selected: bbq,ranch.
Sides selected: chips.
Salads selected: .
Drink selected: dasani_mineral_water.
============================================================
======================== THANK YOU =========================
============== PLEASE COME AND USE ME AGAIN :D =============
============================================================
true .
```

References:

1. *"Home | SUBWAY.com - Singapore (English)."* https://www.subway.com/en-SG. *Accessed 6 Apr. 2019.*
2. "An introduction to Prolog! - photos.boklm.eu." https://boklm.eu/prolog/page_5.html. Accessed 6 Apr. 2019. (exclamation mark usage explanation)
3. "Eating Healthy at Subway | SparkPeople." https://www.sparkpeople.com/resource/sparkdining-eatery.asp?id=6. Accessed 6 Apr. 2019.
4. "How to Eat Vegan at Subway | PETA." 8 Sep. 2016, https://www.peta.org/living/food/eat-vegan-subway/. Accessed 6 Apr. 2019.
5. "Dynamic Switch Case in Prolog - Stack Overflow." 27 May. 2016, https://stackoverflow.com/questions/37463304/dynamic-switch-case-in-prolog. Accessed 6 Apr. 2019.
6. "SWI-Prolog -- fail/0." http://www.swi-prolog.org/pldoc/man?predicate=fail/0. Accessed 8 Apr. 2019. (Fail predicate explanation)
7. Hanstananda. “Hanstananda/cz3005.” *GitHub*, 8 Apr. 2019, github.com/hanstananda/cz3005. (Github Repository for this project)