Project 1 - Favor for the Ringmaster

CmpE 230, Systems Programming, Spring 2024

By : Efe Feyzi Mantaroğlu, Volkan Bora Seki

In this program we try to build an interpreter just like python. We first parse the sentence and determine actions we need to take.

First, we divide the given input into words using “strtok” function. We also get the word count from it. If the word count is only 1, it should be the exit command, so we check it. After that we look for a question mark at the end of the sentence. If it exists, we use “parse Question” function to get the information. Else we use the “parse” function.

To parse a question, we use word counts. If the word count is 3 (we count “?” as a word.) it can only be “SUBJECT WHERE ?” or “SUBJECT TOTAL ?”. So we check if first word is a subject and check the second word is a where or total keyword. If word count is 4, it can only be “SUBJECT TOTAL ITEM ?” (Items consist of 2 words.) or “WHO AT LOCATION ?” query. We again check the words and return false if some word does not match. If word count is more than 4 it should be “SUBJECT(s) TOTAL ITEM ?” query. We use the “look for” which will be mentioned later to look for subjects. If the function stops at the total keyword, we then look for an item if we found all of them, we return true. If parseQuestion returns true, therefore it is a valid sentence, we add query to the actionList which is also going to be mentioned later.

Parsing non-question sentences is complicated. So, we implemented a function to simplify parsing those. Since every statement can be simplified as SUBJECT(s) ACTION ITEM(s)/LOCATION ADDITIONAL_ACTION SUBJECT. We look for these constructs using “look_for” function. It is a function to search for subject, item or location and stop when it sees a keyword and after that store those in an ItemList struct. ItemList is simply an array with a property “size” to show its length.

For subjects, look_for tries to get as many words concatenated by "and” possible before stopping at a keyword. We do not count and as a keyword for look_for since it is essential for subjects and items. Location logic works the same. It only takes one word.

For items it works similar to subjects, only it takes 2 words at once and concatenates them. It also stops when it sees a keyword. Only difference is that it checks validity while parsing and can go back if function assumed a word as an item even if it is not.

Examples when look_for is called with SUBJECT:

Ali and Ahmet buy 5 bread.

Ali and Ahmet buy 5 bread. (Ali is not a keyword)

Ali and Ahmet buy 5 bread. (Always take and)

Ali and Ahmet buy 5 bread. (Ahmet is not a keyword)

Ali and Ahmet buy 5 bread. (Stop, since buy is a keyword, add “Ali”, ”Ahmet” to ItemList, return with keyword “BUY”)

Ali and Ahmet and

Ali and Ahmet and (Ali is not a keyword)

Ali and Ahmet and (Always take and)

Ali and Ahmet and. (Ahmet is not a keyword)

Ali and Ahmet and (Always take "and").

Ali and Ahmet and (Stop because there are no other words to parse. Since this is not a valid SUBJECT(s) construct, return -1)

Examples when look_for is called with ITEM after buy:

Ali buy 2 apple and 1 banana

Ali buy 2 apple and 1 banana (Take items)

Ali buy 2 apple and 1 banana (Always take and)

Ali buy 2 apple and 1 banana (Take items, add items to list and return with "END_OF_STRING" keyword)

Ali buy 2 apple from Ahmet

Ali buy 2 apple from Ahmet (Take items and stop when there is a keyword. Return with "FROM" keyword.)

Ali buy 2 apple and Ahmet go to Konya

Ali buy 2 apple and Ahmet go to Konya (Take items)

Ali buy 2 apple and Ahmet go to Konya (Take and)

Ali buy 2 apple and Ahmet go to Konya (Take items)

Ali buy 2 apple and Ahmet go to Konya (Since the last item is not really an item go back to last item and return the function with "NEXT_SEQUENCE")

So, to parse a sentence we use parse function, and it utilizes look_for function. Since all sentences in the homework begin with a SUBJECT, we first call look_for with a SUBJECT call. We expect function to return BUY, SELL or GO keyword. If it returns a keyword other than that we return false. After that we use look_for according to keyword. If we stopped at BUY keyword we check for items if we see "from" while checking items, we check for subjects and so on.

Parse function is also recursive. Even though the sentence we checked is valid, another sentence might be invalid. So, whenever we get "NEXT_SEQUENCE" keyword from look_for we call parse function again with the remaining sentence and check if it is valid. This way we can detect errors early.

Examples of parse function:

Ali and Ahmet buy 5 bread and Ahmet go to Konya

Ali and Ahmet buy 5 bread and Ahmet go to Konya (look_for : SUBJECT returns “BUY”)

Ali and Ahmet buy 5 bread and Ahmet go to Konya (look_for : ITEM returns “NEXT_SEQUENCE”)

Add action to action list : (BUY,[”Ali”, ”Ahmet”],[“5”,”bread”],NULL)

Ali and Ahmet buy 5 bread and Ahmet go to Konya (Call parse function again)

Ahmet go to Konya (look_for : SUBJECT returns “GO”)

Ahmet go to Konya (look_for : LOCATION returns “END_OF_STRING”)

Add action to action list : (GO,[”Ahmet”],[”Konya”],NULL)

Since both of the sentences are valid, return true.

Ali and Ahmet buy 5 apple from Efe and Volkan

Ali and Ahmet buy 5 apple from Efe and Volkan (look_for : SUBJECT returns “BUY”)

Ali and Ahmet buy 5 apple from Efe and Volkan (look_for : ITEM returns “FROM”)

Ali and Ahmet buy 5 apple from Efe and Volkan (look_for : LOCATION returns “NEXT_SEQUENCE") (We use location because there can only be 1 seller.)

Add action to action list : (FROM,[“Ali”, ”Ahmet”],[”5”, ”apple”],[“Efe”])

Ali and Ahmet buy 5 apple from Efe and Volkan (call parse function again)

Volkan (look_for : SUBJECT returns “END_OF_STRING”)

This returns false because sentence does not have any action. So, parse functions return false.

Look_for function also stops at if statements since it is a keyword. After seeing if keyword we call parseIf function. It works really similar to parse function. Actually, it runs exactly same as parse function, only difference is it looks for has or at keywords instead of buy or sell. At first, we thought that if statement affects only the last action. So, there was a recursive and complicated parseIf function. However, we later learned that it affects all actions so it works very similar to parse function.

Examples of parse and parseIf functions combined:

Efe and Ahmet if

Efe and Ahmet if (Parse function, look_for : SUBJECT returns IF keyword which is invalid.)

Efe buy 5 apple if Ahmet at Konya

Efe buy 5 apple if Ahmet at Konya (Parse function, look_for : SUBJECT returns “BUY”)

Efe buy 5 apple if Ahmet at Konya (Parse function, look_for : ITEM returns “IF”)

Add action to action list : (BUY,[“Efe”],[”5”, ”apple”],NULL)

Efe buy 5 apple if Ahmet at Konya (Call for parseIf)

Ahmet at Konya (ParseIf, look_for : SUBJECT returns “AT”)

Ahmet at Konya (ParseIf, look_for : LOCATION returns “END_OF_STRING”)

Add action to action list : (AT , [“Ahmet”] , [”Konya”] , NULL)

Returns true.

If parse function returns false, program prints “INVALID”. Else program executes actions.

If parse function returns true, there will be a sentence array. Sentence array is an array of actions and conditions. For example,

[(BUY,[“Efe”],[”5”, ”apple”],NULL),
(FROM,[“Ali”, ”Ahmet”],[”5”,”apple”],[“Efe”]),
(AT,[“Efe”],[“Konya”],NULL),
(HAS,[“Efe”],[“3”,”bread”],NULL),
(SELL,[“Volkan”],[“2”,”sword”],NULL),
(AT,[“Bora”],[“Mersin”],NULL)]

This sentence is a complex sentence in which some actions are dependent on some conditions and others are dependent on other conditions. The program separates the sentence according to conditions and actions which are dependent on the conditions.

There is a function traverses the array and returns array of integer arrays. Every integer array has 3 defined integer.
[1_1 , 1_2 , 1_3], [2_1 , 2_2 , 2_3], …
1_1: the start of actions of the first sentence.
1_2: the start of conditions of the first sentence

1_3: the end of conditions of the first sentence

2_1: the start of actions of the second sentence.
2_2: the start of conditions of the second sentence

2_3: the end of conditions of the second sentence
…
Automatically the traverse function makes the action array the length of input (in this case 6). On the other hand, there exist only 2 sentences. The code remembers the number of separate sentences.
With a for loop, function first checks the conditions. If x_2’th action is not a condition, just do the actions from x_1 to end. Otherwise, the code checks the conditions from x_2 to x_3 and if all of them are true, code do the actions from x_1 to x_2.

I also want to examine the action and “ifcheck” function:

Action:
code: An integer represents the action to be performed.
subjectList: the subjects involved in the action.

objectList: the objects involved in the action.

othersList: additional objects involved in the action.

If action does not require another term otherslist is NULL.
The code of actions

BUY: Buys items for the subjects from the objects list.

SELL: Sells items for the subjects from the objects list.

BUY_FROM: Buys items for the subjects from the objects list and sells them to others from the others list. Others list is not NULL this time.

SELL_TO: Sells items from the subjects to the objects and buys items for others. Others list is not NULL this time.

GO: Moves the subjects to the specified location.

ifCheck:
question: An integer representing the question to be checked.

subjectlist: subjects involved in the question.

Objectlist: objects involved in the question.

otherlist: additional objects involved in the question.

Question Codes:

AT: Checks if the subjects are at the specified location

HAS: Checks if the subjects have the specified quantity of the specified item from the objectlist.

MORE: Checks if the subjects have more than the specified quantity of the specified item from the objectlist.

LESS: Checks if the subjects have less than the specified quantity of the specified item from the objectlist.

Code does this for all sentences and returns OK if the sentence is an action-condition sentence.
Sentence may only be a question sentence. Then if the sentence is valid, code returns the answer of the question. Questions are answered in action functions
questions:

TOTAL: Prints the total quantity of a specified item across all subjects.
WHERE: Prints the current location of the subject.

INVENTORY: Prints the current inventory of the subject.
WHO: Prints the people present at the current location.
Code continues to take input from user until "exit" input is given.

Efe Feyzi Mantaroğlu	Volkan Bora Seki
2021400027	2021400156