BLG 311E Term Project - Phase 1

For this first phase of the project, you'll implement a general Context Free Grammar (CFG) parser in **C**. Given a grammar (in **grammar.txt** file) and a list of words (in **words.txt** file), your program will decide whether the given words are in the language defined by the grammar. Note that we don't impose any restrictions on the form of the grammar, so your program should work with any valid CFG. You can also think about this project as a Push-Down Automata (PDA) simulator. Since PDAs are nondeterministic in general, you should think of how to handle nondeterminism. In other words, whenever there are more than one possible rules to apply, you can try to simulate all possible paths to find a working generation (using backtracking whenever a path fails). Be careful, to be able to parse any given CFG without a normal-form is not that easy. You'll notice that for some grammars, blind application of backtracking does not work (see left recursive grammars). In this phase, we don't require you to be able to parse all grammars but you should do your best. At the end, you will submit a **one page report** (report.pdf file) explaining your algorithm and the limitations of your approach. You should explain which grammars your program can not parse.

Specifications

Input:

A sample **grammar.txt** file is given below. First line states the terminals (alphabet) of the language. Second line gives the Non-terminals of the grammar. Note that first non-terminal is the **start symbol.** After these initial two lines, rules of the grammars are given (one rule on each line). Each line starts with the ID of the rule (an integer). Every term in the grammar file is separated by a single space character. For simplicity, you can assume that all the terminals and non-terminals will be just a single character.

```
) ( + 0 1 *
S E
1 S -> ( E )
2 S -> E + E
3 S -> E * E
4 E -> S
5 E -> 0
6 E -> 1
```

The **words.txt** file will contain words (one word on each line). An example words.txt file is given below.

```
(0+1)+1
(1+11)
0*(1)
```

Output:

For each word in the words.txt file, if the word cannot be generated by the grammar you will only print NO. If the word is in the language, you should print YES in the first line and on the next line you will print the rules you applied to generate the word with a **left-most derivation**. The results should be printed to **results.txt** file. For the grammar and words file given in the previous section, your results.txt file should be as follows:

```
YES
2 4 1 4 2 5 6 6
NO
YES
3 5 4 1 6
```

Regulations

- You have to implement in C.
- You will submit two files. One is your parser in file **parser.c** and the other is the report in pdf format, **report.pdf** .
- You are not allowed to use any special parsing libraries.
- You can do the project in groups (at most 3 person). You have to send an e-mail to İsmail Bilgen (ibilgen@itu.edu.tr) with subject [BLG311E GROUP] and the group members' names and IDs in the e-mail.
- Don't try to implement well-known parsing algorithms. We want your best approach.
- Do not share anything in the code level with your friends. Any unnecessary similarities between the codes will receive our special attention. If we find out that you have some piece of code that is not result of your own coding (from friends or from internet without any reference in your report), not only you'll get a 0 from this project but also will lead to disciplinary action. Worst of all, you'll be stamped as a cheater and a fraud.