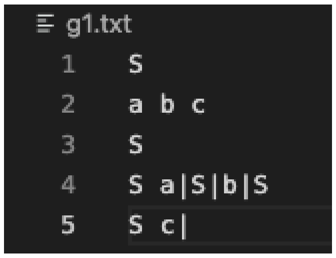
Recursive Descendent Parser

https://github.com/DanBesu/formal-languages-and-computer-design/tree/main/lab567-complete

Example:



g1.txt

```
S
a b c
S
S a|S|b|S
S c|
```

Here, the productions are split by "|", so the productions for non-terminal S will be:

S -> a S b S

S -> a S

S -> c

Class Grammar:

Props:

1. N (the list of non terminals)
2. E (the list of terminals)
3. S (starting symbol)
4. P (productions - dictionary that uses a string as key and a list of lists of symbols from the right side of the production as value)
5. Grammar (a list of lists)
6. Filename (string)

Methods:

1. read_grammar() - read the grammar from the text file
2. represent_productions() - build the P dictionary
3. get_terminals()
4. get_non_terminals()5. get_productions()
6. get_productions_for_non_terminal()
7. print_productions_for_non_terminal()
8. get_start_symbol()

Class Recursive Descendent Parser:

Props:

1. sequence (a list of codes)
2. Grammar
3. input_stack (a list used as stack)

4. working_stack (a list used as a stack)
5. output_file (string)
6. state (string)
7. index (integer)
8. tree (list)

Methods:

1. expand()
2. advance()
3. momentary_insuccess()
4. back()
5. another_try()
6. success()
7. write_in_output_file()
8. write_all_data() (append the state, index, contents of input and working stacks)
9. print_working_stack()
10. get_length_depth()
11. create_parsing_tree()
12. run() (used as “main” function checking if the sequence is correct)
13. write_parsing_tree() (used from the ParserOutput class)
14. read_sequence() (construct the list of codes)
15. init_output_file()