Github link: https://github.com/Matyas2003/FLCD-Lab/tree/main/Lab7

Class Node:
- value = string
- children = list
- parent = None/Node
- index = integer

Class ParserOutput:
- prod = string
- root = Node

function create_tree(self):

Having the string of production resulted from the parser, it iterates through it and builds the tree needed for the parent-sibling table. The string of production will have non-terminals followed by their index (nr of production) and terminals. The tree is built in a preorder manner.

Class Grammar:
- nonterminals = list
- terminals = list
- start_symbol = string
- productions = dictionary
- cfg = boolean

function read_from_file(self, file):

Reads the grammar from the specified file and checks if the grammar is context free.

function print_nonterminals(self), function print_terminals(self), function print_start_symbol(self), function print_productions(self): Helper functions that print the grammar.

function print_prod_for_nonterm(self, nonterm):

Prints productions for a specified non-terminal.

function cfg_check(self):

Prints whether the grammar is context free or not.

Other functions:
function print_tree(root):

Given the root of the tree, prints the nodes in pre-order.

function create_parent_sibling_table(root):

Given the root of the tree and using a queue, it traverses the tree in level order, building the table. In the queue, we keep the tuples that are later placed in the table (node, index of parent, index of left sibling). The way the tuples are popped out of the queue is in level order.

function beta_head(beta):

Returns the head of the input stack.

function alpha_read(alpha):

Return the head of the working stack.

function is_empty(beta):

Returns True if the input stack is not empty and False otherwise.

function expand(config):

If the head of the input stack is a non-terminal, adds the non-terminal followed by its production index to the working stack and adds the production of the non-terminal to the input stack.

function advance(config):

If the head of the input stack is a terminal corresponding to the current symbol from the input, the index is incremented and the terminal moves to the working stack.

function momentary_insuccess(config):

If the head of the input stack is a terminal not corresponding to the current symbol of the input, the state is changed to “back state”.

function back(config):

If the head of the working stack is a terminal, the index decrements and the terminal moves to the input stack.

function another_try(config):

If the head of the working stack is a non-terminal, there are 3 cases:

1. The index is 1 and the head of the working stack is the starting symbol -> changes the state to “error state”.
2. If we have not reached the end of the list of productions for the non-terminal -> changes the state to “normal state”, the index of the non-terminal increases and the head of the input stack is the production found in the list of the non-terminal on the increased index.
3. Otherwise -> the head of the working stack is moved to the head of the input stack without its index.

function success(config):

Changes the current state to “final state”.

function print_result_table(table):

Prints the resulting parent-sibling table on the screen.

function write_table_to_file(table):

Write the resulting parent-sibling table in the “out1.txt” file.

function build_string_of_production(prod):

Given the resulting production, calls functions responsible for creating the tree, creating the parent-sibling table and the ones for printing the table.

function rd_parser():

Main algorithm for parsing the sequence in a recursive descendant method.

function read_sequence():

Reads a sequence from the “seq.txt” file.