```
#!usr/bin/env python3
# FILE: LL_parser.py
# FINAL PROJECT
# Professor Ahmadnia
# Group: Kevin Vuong, Anika Corpus, Christopher Grant
# Description: This program provides a function to parse a tokenized list of input string to
 check
#              whether the input string is a valid string based on the Predictive Parsing
table.

import re

from token import *


def syntax_error_handler_1(error_value, line_number):
    """
    Handles the error condition in the terminal if-block

    :param error_value: The symbol under question
    :param line_number: The line number where the error has occurred
    :return:
    """
    if re.match(r':', error_value):
        print('Line ' + str(line_number) + ':', 'Missing a colon (:)')
    elif re.match(r';', error_value):
        print('Line ' + str(line_number - 1) + ':', 'Expected a semicolon')
    elif re.match(r'\)', error_value):
        print('Line ' + str(line_number) + ':', 'Expected a )')


def syntax_error_handler_2(error_value, line_number):
    """
    Handles the error condition in the non-terminal if-block

    :param error_value: The symbol under question
    :param line_number: The line number where the error has occurred
    :return: None
    """
    error_line = 'Line ' + str(line_number - 1) + ':'

    if re.match(r'(P|Q|R|S|BEGIN)', error_value):
        print(error_line, 'Missing a semicolon (;)')
    elif re.match(r';', error_value):
        print('Line ' + str(line_number) + ':', 'Missing an expression')
    elif re.match(r'\)', error_value):
        print('Line ' + str(line_number) + ':', 'Invalid expression')


def predictive_parser(token_list, predict_table, terminal_list, starting_symbol):
    """
    Determines whether the input string is accepted or rejected based on the prediction
table.

    :param token_list: a list of tokens to parse (basically, the input string in tokenized
form)
    :param predict_table: the prediction table being used
    :param terminal_list: a list of terminals for the grammar
    :param starting_symbol: the symbol to which the grammar starts with
    :return: Returns -1 if input string is rejected, otherwise returns 0 if accepted
    """

    stack = ['$', starting_symbol]  # Push the end-of-input symbol and the starting symbol
    i = 0  # Keeps track which token is currently being read

    # Add ending symbol to the end of the input
    ending_symbol = Token('$', '$', 0)
    token_list.append(ending_symbol)

    while stack:  # loop until stack is empty
```

```
        top_of_stack = stack[len(stack) - 1]
        token_read = token_list[i]
        char_read = token_read.get_value()  # Gets the actual terminal of the token

        if top_of_stack in terminal_list:  # Terminal
            if top_of_stack == char_read:
                stack.pop()
                i = i + 1
            else:
                print('\n1: The grammar has rejected the input string')
                print(top_of_stack, char_read)
                syntax_error_handler_1(top_of_stack, token_read.get_line_num())
                return -1
        else:   # Non-terminal
            if predict_table[top_of_stack][char_read] is not 'aaa':  # If table entry is
not an empty entry
                entry = stack.pop()
                if predict_table[entry][char_read] is not 'lambda':
                    # Push the entry into the stack in reverse order
                    for symbol in reversed(predict_table[entry][char_read].split()):
                        if re.match(r'PROGRAM|BEGIN|END\.|INTEGER|PRINT', symbol):
                            stack.append(symbol)
                        else:
                            for non_terminal in reversed(symbol):
                                stack.append(non_terminal)
            else:
                print('2: The grammar has rejected the input string')
                print(top_of_stack, char_read)
                syntax_error_handler_2(char_read, token_read.get_line_num())
                return -1
        print(stack)

    print('The grammar has accepted the input string\n')
    return True
```