Aisha Siddiqa - i221281
Ayesha Ejaz - i220899

Parsing Strings Using LL(1) Parsing Table

Goals

- Extend the work from Assignment 2 where we created a Context-Free Grammar (CFG) and constructed its LL(1) parsing table.
- **Use** the previously built parsing table to **parse multiple input strings**.
- **Report each parsing step**, identify parsing errors if any, and verify correctness.

Approach

Our approach followed these steps:

Aisha Siddiqa - i221281
Ayesha Ejaz - i220899

1. **Reuse Assignment 2 Code**:
   Implemented new code in the previous code that :
   - ➢ Read a CFG.
   - ➢ Remove Left factoring and Left recursion.
   - ➢ Computes FIRST and FOLLOW sets.
   - ➢ Builds the LL(1) Parsing Table.
2. **Parsing Input Strings**:
   - ➢ We extended the program to read **multiple lines** from an input file.
     - ■ Each line represents a separate input string (tokenized based on spaces).
   - ➢ Each input string was **appended with an end-marker $** to indicate the end of the input.
3. **Parsing Mechanism**:
   - ➢ A **stack-based parser** was implemented.
   - ➢ Initially, the stack is loaded with $ and the **start symbol** of the CFG.
   - ➢ At every step:
     - ■ If the top of the stack **matches** the current input symbol, they are both popped/moved forward.
     - ■ Otherwise:
       - ● If the top is a **non-terminal**, the appropriate production is **expanded** based on the LL(1) parsing table.
       - ● If no matching production exists, an **error** is reported.

4. **Logging Parsing Steps**:
   - ➢ For every action (MATCH, GENERATE, ERROR), the parser writes:
     - ■ The current input symbol
     - ■ The action taken
     - ■ The current contents of the stack
   - ➢ These logs were saved into SEPARATE FILES(parsing_output_line_<line_number>.txt) for each input line.

Challenges Faced

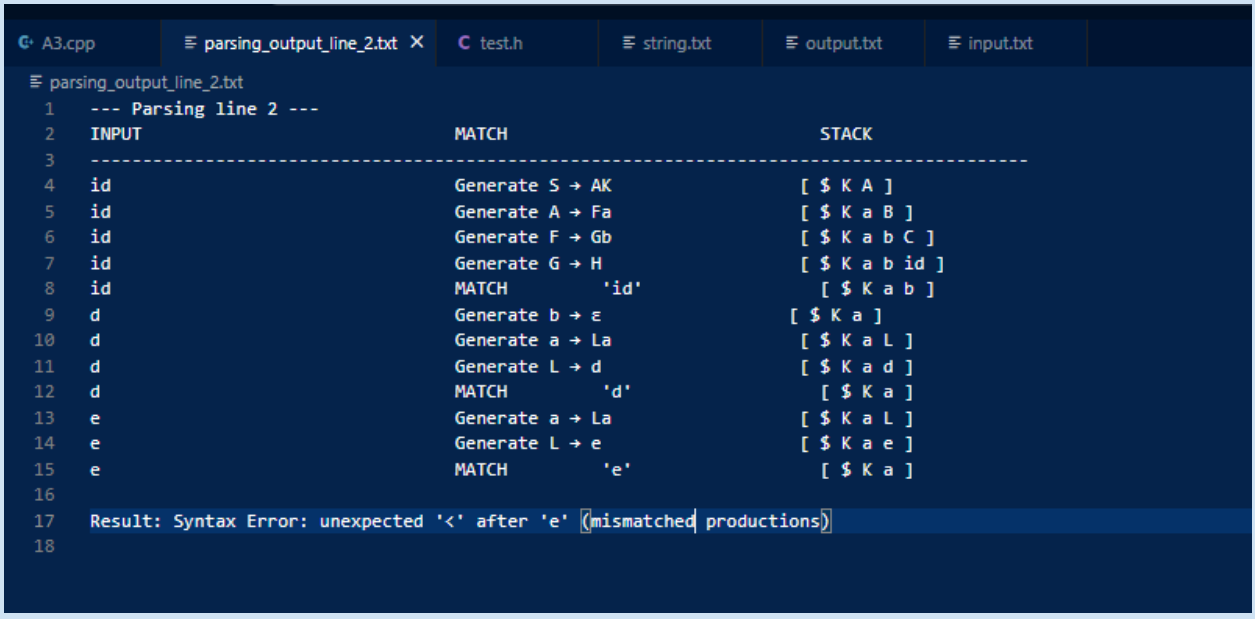
- **Stack Representation and Readability**:
  - ○ While printing the parsing steps, ensuring the **stack contents** looked clean and human-readable was challenging.
  - ○ We developed a helper function stackToString to print the stack from bottom to top in a neat format.

Aisha Siddiqa - i221281
Ayesha Ejaz - i220899

- **Mismatch Between Expected and Actual Input**:
    - One of the difficulties was UNDERSTANDING and clearly explaining different types of syntax errors, such as:
        - Unexpected tokens at the start.
        - Missing symbols mid-parsing.
        - Premature end-of-input errors.
    - Error messages were designed carefully to guide users toward the exact mistake.
    - The given Assignment file *doesn't specify the type of errors.*

- **Using the previous code**:
    - We already had a complex implementation in our previous code, and we had to modify/reuse the previous code which made us not want to start the assignment xD.

- **Output Alignment**:
    - Aligning the columns properly when printing MATCH and GENERATE actions was tricky because text width varies if not viewed with a monospaced font.
    - We used formatting functions like setw() and recommended viewing outputs in a monospaced font (e.g., Consolas) for clarity.
    - Still it was unclear so we detected that the stack column has the max length, so we printed it in the end to keep our output aligned. *And we also shifted all our outputs from terminal to .txt files.*



Aisha Siddiqa - i221281
Ayesha Ejaz - i220899

Correctness Verification

To verify the correctness of the parser:

1. **Successful Parsing**:
   - ➢ Input lines that conform to the CFG were **parsed successfully without any errors**.
   - ➢ These cases printed "Result: parsed successfully" in the respective output files.
2. **Error Handling**:
   - ➢ Input strings with syntax errors correctly triggered meaningful error messages.
   - ➢ Different types of parsing errors were identified (unexpected token, missing symbols, unmatched non-terminals, etc.).
   - ➢ ***All possible error cases were tested,*** using multiple inputs. The input.txt file is attached.
3. **Manual Cross-Checking**:
   - ➢ For some test cases, parsing steps were manually traced alongside the program's output to ensure every stack operation and match was correct.
4. **Summary Output**:
   - ➢ At the end of parsing, the program printed a summary:
     - Total number of input lines
     - Number of successfully parsed lines
     - Number of errors encountered
     - Specific error details for each line

```
Parsing completed.
Total lines parsed: 5
Successful parses: 1
Errors encountered: 4
Line2: Syntax Error: unexpected '<' after 'e' (no required productions)
Line3: Syntax Error: unexpected end of input, expected 'else' (missing symbols)
Line4: Syntax Error: unexpected end of input. (mismatched productions)
Line5: Syntax Error: unexpected '<' after 'if' (no required productions)
aisha@DESKTOP-7OCHQ16:~$
```

This provided an easy way to check overall performance and identify lines that failed.

Aisha Siddiqa - i221281
Ayesha Ejaz - i220899

Conclusion

phenku >.<

This assignment helped us test our previous assignment code.