Compiler Design and Construction

Assignment 1 – Dynamic buffering and Hardcoded Methodology FALL (2024)

Submission Before: 6:00 PM – 21-10-2024
(Late will be penalty of deduction of 2 absolute marks per day)

CLO: <4>; Bloom Taxonomy Level: <Apply>

Objective:
The goal of this assignment is to implement a hardcoded lexical analyzer using a `switch` statement to detect **comments** in source code. The implementation will include **dynamic buffering**, where the buffer loads characters from the input file in chunks of **4096 bytes**. The lexeme (e.g., a comment) will be formed by reading characters from the buffer.

Requirements:

1. **Lexical Analyzer Using Switch**:
   - Implement a lexical analyzer that uses a `switch` statement to detect different types of comments.
   - The analyzer should recognize:
     - Single-line comments (e.g., `// this is a comment`)
     - Multi-line comments (e.g., `/* this is a multi-line comment */`)
2. **Dynamic Buffer for File Loading**:
   - The buffer will dynamically load data from a file in chunks of 4096 bytes.
   - Use a linked list of buffer blocks. Each buffer block represents a 4096-byte segment loaded from the file.
   - The analyzer reads characters from the buffer to form a lexeme (e.g., a comment). If the current buffer block is exhausted (or the end of the buffer is reached), the next buffer block is loaded, and reading continues.
   - **Buffer Block Size**: Each buffer block should have a size of 4096 bytes.
3. **Lexeme Formation**:
   - The lexeme (e.g., a comment) is formed by reading characters dynamically from the buffer, starting from an index `start` and continuing until the lexeme is completed (`end`).
   - For single-line comments, stop at the newline character (`\n`).
   - For multi-line comments, stop at the `*/` token.
4. **Switch-Based State Machine**:
   - Use a `switch` statement to handle different states of the lexical analyzer:
     - **Initial state**: Start reading characters from the buffer.
     - **Comment state**: Detect `//` for single-line comments and `/*` for multi-line comments. Once in the comment state, keep reading until the end of the comment is found (`\n` for single-line or `*/` for multi-line).

    - Continue reading from the current buffer block. If the end of the block is reached, load the next buffer block and continue processing.
5. **Linked List Buffer**:
    - Implement a **linked list of buffer blocks**. Each buffer block should:
        - Have a fixed-size array of **4096 bytes** for storing characters loaded from the file.
        - Contain a pointer to the next buffer block.
        - Maintain a current position ($_{pos}$) to track the read progress within the buffer.
    - When the current buffer block is exhausted, move to the next buffer block, and load the next 4096 bytes from the file (if any).
6. **File Input**:
    - The analyzer will read input from a file containing source code.
    - Implement file handling to open the file, read its contents, and load it into the dynamic buffer.
    - Handle files of arbitrary size by loading multiple buffer blocks as needed.
7. **Memory Management**:
    - Ensure that buffer blocks are dynamically allocated and deallocated as needed.
    - Free all memory allocated for buffer blocks after processing the file.

Submission Requirements:

- Submit the source code implementing the lexical analyzer with dynamic buffering.
- Your program should process an input file and detect all single-line and multi-line comments.
- Ensure that the program handles files of arbitrary size by dynamically loading data into the buffer.
- Include a description of how the dynamic buffer works and how the state machine processes characters to detect comments.