CMSC 430 — Project 2 Final Report

Overview

Project 2 asked us to extend last-term's scanner with a Bison parser that recognises functions, variable declarations, control-flow statements (`WHEN` , `IF/ELSIF/ELSE` , `SWITCH/CASE` , `FOLD`), arithmetic and boolean expressions, and produces clear lexical / syntax / semantic error counts at the end of each compilation run.

My work proceeded in two distinct iterations; the second replaced the first and forms the basis of the final submission.

2. Approach #1 *(discarded)*

Step	Goal	Outcome
1	Follow "Project 2 Requirement.pdf" literally.	Implemented a direct translation of the BNF in the hand-out.
2	Break tasks into a linear to-do list.	Helped at first, but quickly diverged into real-world debugging order.
3	Deep dive into EBNF ⇄ BNF conversions.	Learned a lot of theory, but grammar slowly drifted from Flex token stream.
4	Debug resulting parser.	Consumed most of the time; produced *one* working construct but many shift/reduce conflicts.

Lesson: theory ≠ practice. "Perfect" EBNF conversions are useless if the lexer and parser do not share the same vocabulary and precedence.

3. Approach #2 *(final)*

Key Change	Reason	Result
Started from a minimal, compiling skeleton.	Guarantees one working path through the grammar at every commit.	Zero-conflict starting point.
Treated each *test file* as a small, self-contained spec.	Forces vertical slice debugging (scanner→parser→listing).	Faster feedback; no hidden regressions.
Added **one** new construct, wrote tests, then refactored.	Keeps diff small; easier to bisect.	All constructs stable before next feature.
Stored every green commit under `vX_grammar.y` .	Lets me diff “known-good” snapshots against new bugs.	Reduced re-debug time drastically.
Used `%left / %right / %nonassoc` aggressively.	Eliminates most shift/reduce conflicts up-front.	Final grammar compiles with `%expect 0` .

---

4. Resolved Problems (chronological)

1. **Relational vs. case-arrow token clash** → distinct rules for `>=` `<=` `<>` (**RELOP**) vs. `=>` (**ARROW**).
2. **Colon shadowed by ** → explicit `":" { return ':'; }` before the catch-all.
3. **No ** → added precedence block; removed `relation: expression` ambiguity.
4. **Bare comparisons rejected** → merged `expression RELOP expression` into main `expression` chain.
5. ** allowed only ternary form** → added simple `WHEN condition ';'` alternative.
6. **Stray empty rule in ** → deleted extra `|` .
7. **Multi-var declarations rejected** → made `variable_declarations` left-recursive.
8. **Comma-separated parameters** → `parameters: parameters ',' parameter | parameter | /*empty*/` .
9. **Dangling-ELSIF conflicts** → recursive `ELSIF_declaration` and matched/unmatched split.

10. **Counters all incrementing at once** → `appendError()` now bumps exactly one of `lexicalErrors / syntaxErrors / semanticErrors`.

---

5 Test Plan

---

see file called Test Plan

---

6. Lessons Learned

- I learned how Bison integrates with Flex to create a complete parser and lexer system.
- I gained practical experience working with makefiles and how they streamline the build process.
- I developed a better understanding of the different sections and purposes of `.y` (Bison) and `.l` (Flex) files.
- I improved my knowledge of user-defined functions and how to add them to both the parser and lexer files.
- While I do not consider myself an expert in LR or RR production rules, my grasp of grammar structures and parsing flow has noticeably improved.
- I learned how to incorporate function calls within grammar rules and the implications for parsing.
- Through debugging `parser.y`, I deepened my understanding of how definitions in `scanner.l` interact with the parser.

- I realized that my understanding of error reporting—especially distinguishing between different error categories—remains incomplete. Despite investing significant time in improving error output for both Project 1 and `parser.y`, I was unable to fully resolve separate error category printouts. This is an area where I plan to seek further clarification and learning.