Compiler Construction

Assignment 04

Prepared By:
Aalyan Raza (22i-0833)
Hadiya Tanveer (22i-1332)

Date: 11th May, 2025

Table of Contents

Introduction

For Compiler Construction Assignment 4, we developed a JSON parser and CSV generator using **Flex** (lexer), **Bison** (parser), and **C** for backend logic. The system reads JSON input, constructs an Abstract Syntax Tree (AST), and outputs CSV files that represent nested structures in the JSON. Each nested object or array is written into its own CSV file, with **foreign key references** to maintain hierarchy. A key requirement was to ensure each CSV file starts its `id` column from **1** and maintains **unique IDs** within that file.

This report details how we handled ID generation per table, managed foreign key integrity, and ensured scalability while keeping integration smooth with the existing codebase.

Challenges

1. Table-Specific ID Management

Each table required an independent ID counter starting at 1. Without it, IDs could overlap or break the requirement.

2. Foreign Key Integrity

Nested objects and arrays had to reference their parent table's ID using foreign keys like main_id.

3. Scalability for Complex JSON

The system had to handle multi-level JSON inputs without excessive memory or complexity.

4. Seamless Integration

The ID logic had to integrate cleanly with existing components: the lexer (scanner.l), parser (parser.y), and AST (ast.c, ast.h).

Solution

We implemented a **table-specific ID system** in `csvgen.c`. Each table's `id_counter` starts at 1 and increments per row added. This logic preserves foreign key relationships and fits cleanly into the CSV generation logic.

Key Implementation Details

1. Modified `Table` Struct

```
typedef struct Table
{
    char name[64];
    FILE *fp;
    int header_written;
    int id_counter;  // Track table-specific ID
} Table;
```

2. Initialized `id_counter` in `get_table`

```
t->id_counter = 1;
```

3. Assigned IDs in `process_object`

```
int id = t->id_counter++;
```

4. Writing Rows in `write_row`

```
fprintf(t->fp, "%d", id);
```

Example Output

For the JSON:

```
     {

      "name": "Alice",
      "address": { "street": "123 Main St", "city":
"Boston" },
      "orders": [
        { "item": "Book", "price": 10 },
        { "item": "Pen", "price": 2 }
      ]
    }
```

Generated CSVs:

Main.csv

```
id,name
1,Alice
```

Address.csv

```
id,main_id,street,city
1,1,123 Main St,Boston
```

Orders.csv

```
id,main_id,item,price
1,1,Book,10
2,1,Pen,2
```

Each CSV starts from ID 1 and maintains correct foreign key references.

Additional Considerations

- **Memory Management:**

  free_ast(root) in main.c deallocates memory correctly.

- **Error Handling:**

  `fopen` errors are caught and reported.

- **Testing**

  We verified functionality using deeply nested JSON and checked CSV outputs for:
  - Unique per-table IDs
  - Foreign key consistency
  - Clean AST structure via --print-ast

- **Minimal Integration Changes:**

  The ID system was added without breaking lexer/parser/AST functionality.

Conclusion

This assignment demonstrates how Flex, Bison, and C can be used to build a robust JSON-to-CSV compiler. By introducing **per-table ID counters**, we maintained ID uniqueness and foreign key integrity across multiple CSVs. The solution is modular, scalable, and easily extendable.
