A Report on Compiler Design Lab (CS304) Mini Project

Mohnish Hemant Kumar (Roll No: 231CS235)

Nikhil Kottoli (Roll No: 231CS236)

Pal Patel (Roll No: 231CS240)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALURU-575025
13-August-2025

Contents

1 Overview of the Code

The provided Lex file implements a lexical analyzer (scanner) for a subset of the C programming language. It uses Flex to define rules for recognizing tokens such as keywords, types, identifiers, numbers, strings, character literals, operators, punctuators, and preprocessor directives. The scanner also maintains a symbol table and constant table. The output is written to `output.md`.

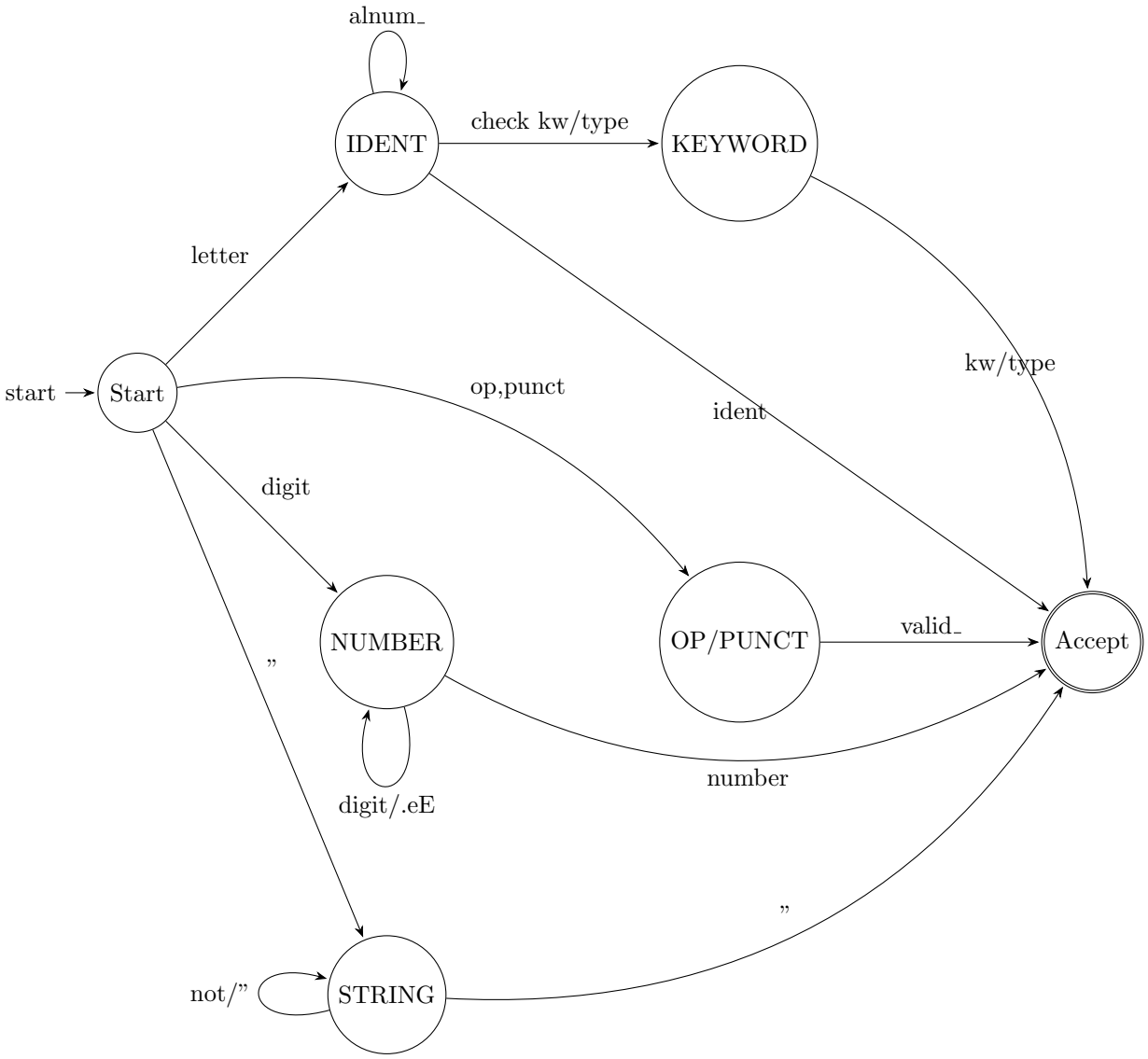
2 Code

Github: `https://github.com/NikhilKottoli/Compiler-Design`

3 List of Recognized Tokens and Their Meaning

- **PREPROC**: Preprocessor directives starting with #.
- **KEYWORD**: Reserved words like `if`, `for`, `while`, etc.
- **TYPE**: Data types like `int`, `float`, etc.
- **IDENT**: Identifiers.
- **NUMBER**: Numeric literals.
- **STRING**: String literals.
- **CHAR**: Character literals.
- **PUNCT**: Punctuation like `( ) { } ;`.
- **OP**: Operators like `+ - * /`.

4 DFA Diagram Underlying the Scanner



5 Assumptions Made Beyond the Basic Language Description

The scanner assumes a C89/C90-like subset, without C99/C11 features like // comments in all contexts (but handles them), or complex numbers. Nested comments are supported (non-standard in C, but code handles depth). Preprocessor only handles #define for constants; other directives are tokenized but not processed further. Identifiers in declarations capture types only on first occurrence; multi-word types (e.g., unsigned int) are concatenated. Function parameters are captured as raw strings (including types), separated by ';' for multiple calls. Array dimensions are appended only for immediate [num] after identifier in declarations. Numeric suffixes (u, l, f, etc.) are recognized but not affecting type beyond classification. No support for trigraphs or digraphs. Errors are reported for unterminated strings/chars/comments and invalid tokens, but scanning continues. Whitespace in array dimensions [num] is allowed.

6 Test Cases with Results, Including Any Failures

6.1 Test Case 1: Simple Program

Input:

```
int main() {
    int a = 10;
    float b = 2.5;
    char c = 'x';
    return a + b;
}
```

Symbol Table (excerpt):

Name	Type	Dimension	Frequency	Return Type	Parameters
main	int	global	1	function	-
a	int	local	2	variable	-
b	float	local	3	variable	-
c	char	local	4	variable	-

Constant Table (excerpt):

Name	Value	Type	Line
-	10	int	2
-	2.5	float	3
-	'x'	char	4

No errors.

6.2 Test Case 2: With Errors

Input:

```
int main() {
    string s = "Hello; // unterminated string
    @illegal = 5;     // invalid token
    return 0;
}
```

Expected Output (excerpt):

- ERROR: Unterminated string literal on Line-2
- ERROR: Invalid token `@` on Line-3

Failures: The scanner recovers by continuing after errors, but may misclassify subsequent tokens if states are not reset properly.

6.3 Test Case 3: Function with Params

Input:

```
#include <stdio.h>
#define SIZE 100
#define PI 3.14159
```

```
int area(int r, char c) {
    int arr[SIZE];
    float result = PI * arr[0] * arr[0];
    return result;
}
```

Symbol Table (excerpt):

Identifier	Type	Scope	Line	Extra Info
SIZE	macro	global	2	100
PI	macro	global	3	3.14159
area	int	global	5	params: (int r, char c)
arr	int[]	local	6	size = SIZE
result	float	local	7	variable

Constant Table (excerpt):

Name	Value	Type	Line
SIZE	100	int	2
PI	3.14159	float	3
-	0	int	8

7 Documentation on Handling Comments, Strings, and Errors

Comments: Single-line: "//" followed by anything until newline; ignored. Multi-line: "/" to "/", with nesting support (depth counter). If unterminated (EOF), reports "ERROR: Unterminated comment" and exits state. Handled in exclusive "comment" state to consume input without tokenizing comment

Strings: Start with ", consume until " (non-escaped), supporting escapes (\a, \b, \xHH, octal) and multi-line via \ at EOL. If newline without " reports ERROR: Unterminated string and resets to INITIAL. Exclusive "string" state with yymore() to accumulate lexeme.string Added to constants as "string".

Errors: Invalid tokens: Any unmatched char triggers "ERROR: Invalid token 'char'". Unterminated constructs: Specific messages for comments, strings, chars. Output to "output.md" with line numbers; scanning continues after errors. No syntax errors (that's parser's job); only lexical.