HW #2 (Lexical Analyzer)

COP 3402: System Software

Fall 2025

Instructor: Jie Lin, Ph.D.

Due date: **Friday, October 3, 2025 at 11:59 PM ET**

Last updated: September 21, 2025

Disclaimer: This document does not cover all possible scenarios. For clarifications, contact the instructor.

Updates: All official updates and clarifications will be posted as **Webcourses announcements**. Check Webcourses regularly for critical updates.

Submission Method: Submit only via Webcourses. Submissions by email, chat/DM, cloud links, or any other channel are not accepted.

Timestamp: The Webcourses submission timestamp is authoritative. All deadlines use U.S. Eastern Time.

Group Work: This is a group assignment.

Contents

1 Academic Integrity, AI Usage, and Late Policy

1.1 AI Usage Disclosure

If you plan to use AI tools while preparing your assignment, you must disclose this usage. Complete the **AI Usage Disclosure Form** provided with this assignment. If you used AI, include a separate markdown file describing:

- The name and version of the AI tool used.
- The dates used and specific parts of the assignment where the AI assisted.
- The prompts you provided and a summary of the AI output.
- How you verified the AI output against other sources and your own understanding.
- Reflections on what you learned from using the AI.

If you did not use any AI, check the appropriate box on the form. **Each team member must submit their own signed AI disclosure form individually in separate submissions (not as a group submission, since group submissions only allow one member to submit and will override other submissions).** Submit the signed form and the markdown file (if applicable) along with your assignment. Failure to disclose AI usage will be treated as academic dishonesty.

1.2 Plagiarism Detection

All submissions will be processed through JPlag, which detects the similarity score between you and the other students' submitted code. If the similarity score is above certain threshold, your code will be considered as plagiarism.

While AI tools may assist with brainstorming or initial code draft (if properly disclosed), the final submission must represent your own work and understanding. It is important to notice that if the similarity score is above the threshold, it does not matter if you have the AI disclosure or not, your program will be considered plagiarism. Also importantly, AI tends to draft the code in a similar way, if you just copy and paste, the similarity score will be very high.

1.3 Late Policy

Due vs. late. Anything submitted after the posted due date/time is late.

Late window. Late submissions are accepted for up to 48 hours after the due date/time; after that, the assignment is missed (score 0).

Penalty (points, not percentages). 5 points are deducted for each started 12-hour block after the due date/time (any fraction counts as a full block):

- 0:00:01–12:00:00 late → -5 points

- 12:00:01–24:00:00 late → -10 points
- 24:00:01–36:00:00 late → -15 points
- 36:00:01–48:00:00 late → -20 points
- After 48:00:00 late → Not accepted; recorded as missed (0 points)

Example: If the assignment is scored out of 100 points, the penalties above are deducted from your earned score.

Technical issues. Individual device/network problems do not justify exceptions—submit early and verify your upload.

2 Assignment Overview

In this assignment, you will implement a lexical analyzer for the programming language PL/0. The lexical analyzer is the first phase of a compiler. It reads the PL/0 source program character by character, groups characters into meaningful lexemes, assigns each lexeme an internal token, and reports any lexical errors. For detailed information on input and output formats with examples, refer to Appendix A.

3 Recognition Rules

The lexical analyzer recognizes the following elements and ignores others:

3.1 Recognized Elements

Reserved words: See Table 1 for the complete list of reserved words.

Table 1: Reserved Words Table

Symbol	Token Symbol	Token Number	Meaning/String Match
begin	beginsym	20	Reserved word
end	endsym	21	Reserved word
if	ifsym	22	Reserved word
fi	fisym	23	Reserved word
then	thensym	24	Reserved word
while	whilesym	25	Reserved word
do	dosym	26	Reserved word
call	callsym	27	Reserved word
const	constsym	28	Reserved word
var	varsym	29	Reserved word
procedure	procsym	30	Reserved word
write	writesym	31	Reserved word
read	readsym	32	Reserved word
else	elsesym	33	Reserved word
even	evensym	34	Reserved word

Special symbols: See Table 2 for the complete list of special symbols.

Table 2: Special Symbols Table

Symbol	Token Symbol	Token Number	Meaning/String Match
+	plussym	4	Addition operator
-	minussym	5	Subtraction operator
*	multsym	6	Multiplication operator
/	slashsym	7	Division operator
=	eqsym	8	Equality operator
<>	neqsym	9	Not equal operator ($\neq$)
<	lessym	10	Less than operator
<=	leqsym	11	Less than or equal ($\leq$)
>	gtrsym	12	Greater than operator
>=	geqsym	13	Greater than or equal ($\geq$)
(	lparentsym	14	Left parenthesis
)	rparentsym	15	Right parenthesis
,	commasym	16	Comma separator
;	semicolonsym	17	Semicolon
.	periodsym	18	Period/dot
:=	becomessym	19	Assignment operator

Identifiers: An identifier begins with a letter and may be followed by letters or digits. The maximum length is eleven characters. See Table 3 for regex representation and token details.

Numbers: A number consists of one or more digits. The maximum length is five digits. See Table 3 for regex representation and token details.

Table 3: Identifier and Number Token Definitions

Token Type	Token Symbol	Token Number	Regex Representation	Constraints
Identifier	identsym	2	`[a-zA-Z][a-zA-Z0-9]*`	Max 11 characters
Number	numbersym	3	`[0-9]+`	Max 5 digits

3.2 Ignored Elements

Comments: Begin with `/*` and end with `*/`. Comments can span multiple lines. During lexical analysis, the lexical analyzer first recognizes and tokenizes the comment delimiters, then excludes all content between these delimiters from the final output. See Table 4 for details.

Table 4: Comment Format Definitions

Start Delimiter	End Delimiter	Description
/*	*/	Block comment

Invisible characters: Tab, whitespace, and newline characters are skipped and not tokenized. See Table 5 for details.

Table 5: Invisible Characters (Ignored During Lexical Analysis)

Character	Description
Space	Space character (ignored)
Tab	Horizontal tab (ignored)
Newline	Line feed character (ignored)
Carriage Return	Carriage return character (ignored)

4 TokenType Enumeration

The following C typedef enum defines all recognized symbols in the PL/0 lexical analyzer. Each token is assigned a unique numeric value corresponding to the token numbers specified in the recognition tables above. This enumeration can be used directly in your C implementation to represent token types.

TokenType Enumeration in C

```c
typedef enum {
    skipsym = 1,     // Skip/ignore token
    identsym,        // Identifier
    numbersym,       // Number
    plussym,         // +
    minussym,        // -
    multsym,         // *
    slashsym,        // /
    eqsym,           // =
    neqsym,          // <>
    lessym,          // <
    leqsym,          // <=
    gtrsym,          // >
    geqsym,          // >=
    lparentsym,      // (
    rparentsym,      // )
    commasym,        // ,
    semicolonsym,    // ;
    periodsym,       // .
    becomessym,      // :=
    beginsym,        // begin
    endsym,          // end
    ifsym,           // if
    fisym,           // fi
    thensym,         // then
    whilesym,        // while
    dosym,           // do
    callsym,         // call
    constsym,        // const
    varsym,          // var
    procsym,         // procedure
    writesym,        // write
    readsym,         // read
    elsesym,         // else
    evensym          // even
} TokenType;
```

5 Lexical Error Reporting

The lexical analyzer must detect and report lexical errors, but does not need to report grammar errors, syntax errors, or semantic errors. Your program should scan through the entire input program and report all lexical errors encountered. The following three types of lexical errors must be detected:

1. **Identifier too long:** Identifiers exceeding 11 characters
2. **Number too long:** Numbers exceeding 5 digits
3. **Invalid symbols:** Characters that are not part of the PL/0 language specification

Error Reporting Strategy: Your program should continue scanning through the entire source program even after encountering lexical errors, collecting all errors before reporting them. This allows the user to see all lexical issues in a single run rather than having to fix errors one at a time.

For examples of lexical error detection and reporting, see Appendix B which demonstrates input files containing lexical errors and their corresponding error output.

6 How to Compile

Compile your scanner on **Eustis**. The only permitted language is **C**. Name your program `lex` and place all of your logic in `lex.c`.

Compilation Commands

```
gcc -O2 -std=c11 -o lex lex.c
```

7 Command Line Parameters

Invoke your program from the terminal on the university grading server **Eustis**. The program must accept **exactly ONE** command-line parameter:

1. *input file name* – the path to the text file containing the PL/0 source program to be scanned.

Do not prompt for input; reject incorrect argument counts. Your scanner should write all output to standard output and should *never* request additional input from the user. If the argument count is not exactly one, print a helpful usage message and exit.

Command Line Usage Examples

Correct usage (program already compiled):

```
./lex InputFile.txt
```

Incorrect usage (wrong number of arguments):

```
./lex
./lex InputFile.txt output.txt
```

Expected behavior with incorrect arguments: The program should print a usage message and exit with an error code.

8 Submission Instructions

Submit on **Webcourses**. Programs are compiled and tested on **Eustis**. Follow these to avoid deductions.

8.1 Code Requirements

- **Program name.** Name your program `lex.c`.
- **Header comment (copy/paste).** Place this non-breaking box at the top of your source file. It will not split across pages and lines will not wrap inside the box.

Required Header Comment

```
/*
Assignment:
lex - Lexical Analyzer for PL/0

Author: <Your Name Here>

Language: C(only)

To Compile:
  gcc  -O2 -std=c11   -o lex lex.c

To Execute (on Eustis):
  ./lex <input file>

where:
  <input file> is the path to the PL/0 source program

Notes:
  - Implement a lexical analyser for the PL/0 language.
  - The program must detect errors such as
    - numbers longer than five digits
    - identifiers longer than eleven characters
    - invalid characters.
  - The output format must exactly match the specification.
  - Tested on Eustis.

Class: COP 3402 - System Software - Fall 2025

Instructor: Dr. Jie Lin

Due Date: Friday, October 3, 2025 at 11:59 PM ET
*/
```

8.2 What to Submit

- Your source code (exactly one of `lex.c`).
- The AI Usage Disclosure Form with your signature.
- **If you used AI:** A separate markdown file describing your AI usage in detail, including the complete dialogue with the AI tool.
- **If you did not use AI:** Only the signed disclosure form is needed.

- The assigned Team Contribution Sheet (must be submitted by each team even you are working alone).

9 Grading

Your assignment will be graded based on the correctness of your lexical analyzer and adherence to the specification. The grading rubric is organized into the following categories with specific point deductions:

9.1 Integrity

- **-100 points:** Plagiarism or resubmission of old programs. **All submissions are processed through JPlag for similarity detection.** If the similarity score exceeds the threshold, submissions will undergo manual verification. Direct copying from other students or direct usage of AI-generated code will result in a score of zero for the entire assignment, regardless of AI disclosure status.

9.2 Compilation & Execution

- **-100 points:** Programs that don't compile on Eustis using the prescribed compilation commands.
- **-100 points:** Program cannot read command line arguments for exactly one argument.
- **-30 points:** Program cannot reproduce any output in the terminal.
- **-10 points:** Program is white-space dependent. This represents a fundamental error in lexical analysis implementation where the program incorrectly relies on specific whitespace arrangements to function properly.
  - **Hint:** For example, `a+b` should be properly tokenized as three separate tokens: identifier `a`, plus operator `+`, and identifier `b`.
  - **Hint:** `4hello` should be tokenized as two tokens: a number `4` and an identifier `hello`.

9.3 Submission Files

- **-100 points:** Missing `lex.c` source file.
- **-5 points:** Missing AI Usage Disclosure Form with signature.
- **-5 points:** Student indicated AI usage but did not submit the detailed dialogue with the AI tool. This will be treated as lack of AI disclosure.

- **-2 points:** Header comment not modified to indicate your name as the author. The required header comment must be personalized with your actual name.

- **-5 points:** Code lacks sufficient comments. Your implementation should include meaningful comments explaining key logic, algorithms, and complex sections.

9.4 Lexical Error Detection

- **-15 points:** Not detecting all three lexical errors (identifier too long, number too long, invalid symbols). Each lexical error detection is worth 5 points.

9.5 Output Formatting

- **-10 points:** Output significantly unaligned or deviates from the specification in Appendix A.

9.6 Code Quality

- **-5 points per issue:** For each unspecified error identified during debugging that prohibits graders from reproducing the expected output. This includes but is not limited to: logic errors, incorrect tokenization patterns, improper error handling, or any implementation flaws that prevent the program from generating the specified output format without modification. Students should thoroughly test their implementations to ensure they work correctly without requiring any manual intervention or code adjustments by graders.

9.7 Plagiarism Detection and Manual Verification

Important Notice: We take academic integrity seriously. All submissions undergo automated similarity analysis through JPlag. If similarity scores exceed established thresholds, the following process will be initiated:

1. **Manual Verification:** Submissions with high similarity scores will be manually reviewed by instructional staff.

2. **Zero Tolerance Policy:** If manual verification reveals direct copying from other students or direct usage of AI-generated code without substantial modification and understanding, the entire assignment will receive a score of zero.

3. **AI Disclosure Irrelevant:** Having an AI disclosure form does not exempt submissions from plagiarism penalties if direct copying is detected.

4. **Similarity Patterns:** AI tools often generate similar code patterns. Simply copying and pasting AI-generated code will likely result in high similarity scores and trigger manual review.

The primary grading criterion is the automated test pass rate combined with adherence to academic integrity standards. Follow all instructions precisely; deviations may result in additional deductions at the graders' discretion.

A Sample Input and Output Examples (No-errors)

A.1 Sample Input Program 1

The following is a sample input file containing a PL/0 program:

Sample Input Program 1 Content

```
var x, y;
begin
y := 3;
x := y + 56;
end.
```

A.2 Output Format Specification

The output format should be printed directly to the console/terminal as standard output. The output consists of three sections in the following order: **Source Program**, **Lexeme Table**, and **Token List**.

Print three sections to standard output (console/terminal). Each section title ("Source Program:", "Lexeme Table:", and "Token List:") should be preceded and followed by a blank line to improve readability. The "Source Program" section should reproduce exactly the contents of the input file. The "Lexeme Table" section must display two columns: the first column contains each lexeme in the order encountered and the second column contains the name of its token type (for example, `varsym`, `identsym`, `;`, etc.). The "Token List" section prints the numeric token codes; for identifiers and numbers you must also include the identifier name or numeric value (in ASCII) next to the token code. Use a consistent delimiter (either a single space or a bar "`|`") between entries and within pairs. Do not insert extra blank lines or interactive prompts.

A.3 Console Output from Sample Input Program 1

This output is the direct output from Sample Input Program 1 when processed by the lexical analyzer. This represents what appears in the console/terminal:

Console Output from Sample Input Program 1

```
Source Program:

var x, y;
begin
y := 3;
x := y + 56;
end.

Lexeme Table:

lexeme  token type
var     29
x       2
,       16
y       2
;       17
begin   20
y       2
:=      19
3       3
;       17
x       2
:=      19
y       2
+       4
56      3
;       17
end     21
.       18

Token List:

29 2 x 16 2 y 17 20 2 y 19 3 3 17 2 x 19 2 y 4 3 56 17 21 18
```

A.4 Sample Input Program 2

The following is a sample input file containing a PL/0 program:

Sample Input Program 2 Content

```
a+b;begin;a,c,c,;
```

A.5 Console Output from Sample Input Program 2

This output is the direct output from Sample Input Program 2 when processed by the lexical analyzer. This represents what appears in the console/terminal:

Console Output from Sample Input Program 2

```
Source Program:

a+b;begin;a,c,c,;

Lexeme Table:

lexeme  token type
a       2
+       4
b       2
;       17
begin   20
;       17
a       2
,       16
c       2
,       16
c       2
,       16
;       17

Token List:

2 a 4 2 b 17 20 17 2 a 16 2 c 16 2 c 16 17
```

B Lexical Error Examples

B.1 Sample Input Program with Lexical Errors

The following is a sample input file containing various lexical errors for demonstration purposes:

Sample Input Program with Lexical Errors

```
begin
    x := 123456;
end.
```

B.2 Console Output for Lexical Error Example

This output demonstrates how lexical errors should be reported. The program scans the entire input and reports all lexical errors found:

Console Output for Lexical Error Example

```
Source Program:

begin
    x := 123456;
end.

Lexeme Table:

lexeme  token type
begin   20
x       2
:=      19
123456  Number too long
;       17
end     21
.       18

Token List:

20 2 x 19 1 17 21 18
```