Department of Computer Science & Engineering

Course: Compiler Design Lab., Course Code: CSC304

Location: NLHC Lab

List of Experiments

Experiment 1

Objective: To implement a combined transition diagram for recognizing a group of tokens.

Brief Theory: Consider the following group of tokens:

a) Identifiers: Letter followed by any number of letter or digit
b) Keywords: BEGIN, END, IF, THEN, ELSE
c) Integer constants: Digit followed by any number of digit
d) Relational operators: <, <=, =, < >, >, >= that are commonly used in any high level language.

Task: WAP to implement the combined transition diagram for recognizing the aforesaid group of tokens.

Apparatus and components required: Computer with C or C++ Compiler and Linux/Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 2

Objective: To design a lexical analyzer for recognizing a group of tokens with the help of flex tools.

Brief Theory: Consider the following specifications of the tokens:

a) Comments are surrounded by /* and */
b) Blanks between tokens are optional, with the exception that keywords must be surrounded by blanks and newlines.
c) Identifier: An identifier is a sequence of letters and digits, starting with a letter. The underscore '_' counts as a letter.
    1. letter → [a-z, A-Z]
    2. digit → [0-9]
    3. id → letter (letter | digit)*
d) Keywords: begin, end, if, then, else, for , do , while, switch, case, default, break, continue, goto

Task: WAP to design a lexical analyser for recognizing a group of tokens with the help of flex tools.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Linux/Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 3

Objective: To identify token of the given definitions.

Brief Theory: Consider the following definitions:

> Special character: "!", "@","#", "$","&", "*","_"
> Token1: Special character followed by any number of letter or digit.
> Token 2: Digit followed by any number of special character or letter.
> Token 3: Start and end with a letter and any number of special character or digit in between.

Task: Write a flex program to identify token of the definitions above.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 4

Objective: To design a lexical analyser for recognizing a group of tokens with the help of flex tools.

Brief Theory: Consider the following specifications of the tokens:

a) Keywords: else, int, void, if, else, while, return. For each one of them, the lexer shall return the tokens INT, CHAR, VOID, IF, ELSE, WHILE, RETURN respectively.
b) It recognizes integer numbers. An integer number is a sequence of digits, possibly starting with a + or -.
c) It recognizes real numbers. A real number is a sequence of digits, possibly starting with a + or – and / or with . and E notations. For each real number, it shall return the token REAL.
d) The lexer shall recognize the operators '->', '&&', '||', '.' for which it shall return the tokens PTR_OP, AND_OP, OR_OP, and DOT_OP respectively.
e) It recognizes operators '-', '+', '*', '/' for which it shall return the same character as token.
f) It recognizes separators ';', '{', '}', ',', '=', '(', ')', '&', '~', , '[' and ']' for which it shall return the same character as token.

Task: WAP to design a lexical analyser for recognizing a group of tokens with the help of flex tools.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Linux/Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 5

Objective: To implement a flex programme to generate strings for the regular expressions.

Brief Theory: Consider the following regular expressions:

a) ((a + b)*(c+d)*)+ + ab*c*d
b) (0 + 1)* + 0*1*
c) (01*2 + 0*2+1)+

Task: Write flex programs for above regular expressions mentioned above.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 6

Objective: To generate table of the precedence functions for an operator/operator precedence grammar.

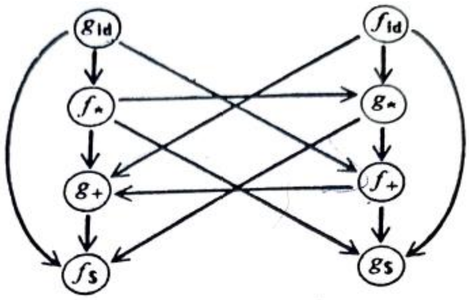
Brief Theory: Given an operator precedence table, we can generate a table of the precedence functions. For example, given the operator precedence table

	id	+	*	$
id		·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	

we can generate the following table of the corresponding precedence functions *f* and *g*

	id	+	*	$
f	4	2	4	0
g	5	1	3	0

by creating a directed graph as follows:



Task: You are **given any** operator precedence table. Write a program to generate the corresponding table of the precedence functions.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 7

Objective: Table generation for an operator precedence grammar.

Brief Theory: Consider the following operator grammar:

E → E+E | E-E | E*E | E/E | E↑E | (E) | id

Task: Write a program to generate the table of operator precedence functions for the above grammar.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 8

Objective: To implement predictive parsing.

Brief Theory: Consider the following grammars:

1.

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

2.

$S \rightarrow A$
$A \rightarrow aB \mid Ad$
$B \rightarrow bBC \mid f$
$C \rightarrow g$

Task: Write a program to design a predictive parser for the above grammar using FIRST and FOLLOW calculations.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 9

Objective: To check whether a given string is palindrome or not.

Brief Theory: A sting is palindrome if it is in the form WcW^T where W^T is the reverse of W where W is formed from the alphabet $\sum = \{a, b\}$.

Task: Write a YACC program to check whether a given string over the alphabet {a, b} is palindrome or not.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 10

Objective: To implement a top-down parsing by recursive procedures.

Brief Theory: Consider the following grammars:

Grammar 1: $S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid f$

Grammar 2:

$S \rightarrow cAd$
$A \rightarrow ab \mid a$

Task: Write two separate programs to implement a top-down parsing by recursive procedures for the grammar.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 11

Objective: Design a shift reduce parser for parsing an arithmetic expression.

Brief Theory: Consider the following grammar:

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid (E) \mid - E \mid id$

Task 1: Write a YACC program to parse an arithmetic expression.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 12

Objective: To implement parsing a string using yacc.

Brief Theory: Consider the following grammar:

S → Aa | bAc |dc | bda

A → d

Task: Write a YACC program to parse if-then-else statement using the grammar above.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 13

Objective: To implement if-then-else grammar.

Brief Theory: Consider the following grammar:

S → iCtS| iCtSeS |a

C →b

Task: Write a YACC program to parse if-then-else statement using the grammar above.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 14

Objective: To implement an LR parser

Brief Theory: Consider the following grammar:

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T*F$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow id$$

Task: Write a program to implement an LR parser for the grammar above and test with a given string.

Apparatus and components required: Computer with C or C++ Compiler and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 15

Objective: To implement error-handler routine for SLR parsing

Brief Theory: Consider the following grammar:

$$E \rightarrow E+E$$
$$E \rightarrow E*E$$
$$E \rightarrow (E)$$
$$E \rightarrow id$$

Task: Write a program for handling error in SLR parsing of the grammar above.

Apparatus and components required: Computer with C or C++ Compiler, Flex, Bison (YACC) and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.

Experiment 16

Objective: To implement syntax directed translation of infix to postfix conversion

Task: Write a program to implement the syntax directed translation of infix to postfix conversion with the usual unary and binary operators.

Apparatus and components required: Computer with C or C++ Compiler, and Windows operating platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.