PROJECT REPORT
ON
COMPILER DESIGN

COURSE NAME: COMPILER DESIGN LABORATORY
COURSE NO: CSE 3212

SUBMITTED TO:

Nazia Jahan Khan Chowdhury
Assistant Professor
Department of Computer Science and Engineering
Khulna University Engineering & Technology, Khulna

Dipannita Biswas
Lecturer
Department of Computer Science and Engineering
Khulna University Engineering & Technology, Khulna

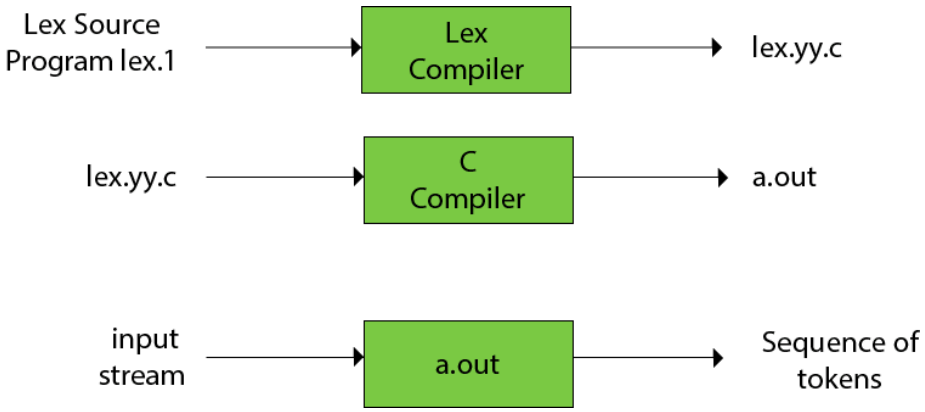
SUBMITTED BY

Md.Fahim Abrar Al Wasi
ROLL: 1907044

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY, KHULNA

FLEX

FLEX (Fast Lexical analyzer generator) is a tool for generating scanners. Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. Scanners perform lexical analysis by dividing the input into meaningful units. For a C program the units are *variables*, *constants, keywords, operators, punctuation* etc. These units are also called as tokens.



BISON

Semantics of a language provide meaning to its constructs, like tokens and syntax structure. Bison is used to perform semantic analysis in a compiler. Bison is a general-purpose parser generator that converts a grammar description for an LALR (1) context-free grammar into a C program to parse that grammar. Parsing involves finding the relationship between input tokens. Bison is upward compatible with Yacc: all properly written Yacc grammar ought to work with Bison with no change. Interfaces with scanner generated by Flex. Scanner called as a subroutine when parser needs the next token.

Flex and Bison are aging Unix utilities that help to write very fast parsers for almost arbitrary file formats. Flex and Bison will generate a parser that is virtually guaranteed to be faster than anything that could be written manually in a reasonable amount of time. Second, updating and fixing Flex and Bison source files is a lot easier than updating and fixing custom parser code. Third, Flex and Bison have mechanisms for error handling and recovery. Finally Flex and Bison

have been around for a long time, so they are far freer from bugs than newer code.

Compiler with Flex and Bison

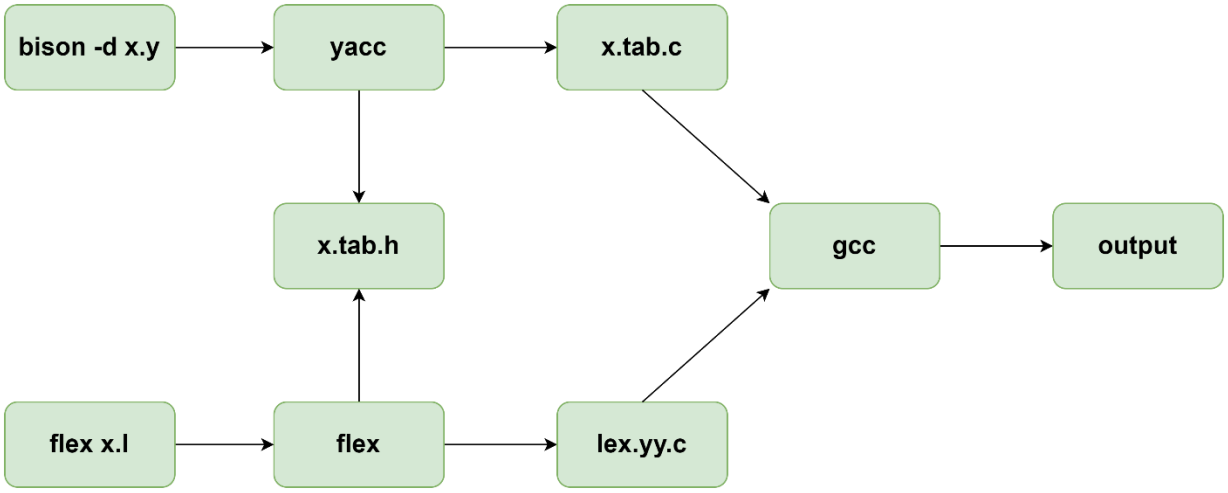


Fig: A diagram of how a compiler build with flex and bison works

Commands to create compiler:

Here *project.y* is the bison file and *project.l* is the lex file.

bison –d project.y

flex project.l

gcc project.tab.c lex.yy.c –o project

After running these commands on command prompt, an executable file named *project.exe* will be created.

Project Description:

Topic	Description
Data Type	Basic data type Int,Float,String Tokens are INTEGR,FLOAT and STRING respectively. RegEx are "numeric","decimal" and "string" respectively.
Variable Declaration	All variables are initialized globally. Syntax: *numeric a,b,c* *decimal y* *string abc*
Variable Initialization	Variable can be initialized while declaring. Syntax: *numeric a<-10,n<-5* *decimal x<-1.5* *string s<-"Hello"*
Value Assign	If the variable is not initialized while declaring it can be assigned value later. Syntax: *&a<-40* *&s<-"World"*

Header	Indicates what header file is included in the code. Syntax: *take_in <stdio.h>* prints stdio.h telling us this header file is included
Comment	Single line comment is handled here .When comment is found it will notify us that this line is a comment Syntax: *#Single line comment*
Print Statement	Prints the data type and value of one or more variavles. Syntax: *display(&a,&b)* *display(&abx)*
Variable Access	The value of a variable can be accessed and manipulate the value to perform different operations. Syntax: To access a variable "&" symbol is used. *&a<-30* The variable a is accessed and 30 is stored inside it.
	Users can define a struct with a variable name and later can create object of this.

Structure	Syntax: *record variable{* *statements* *}* This will show that the struct is defined successfully.
Function	The function will have a name with a return type.This can take parameters.It is not fixed that parameter has to be passed to this function. Syntax: *none method variable(numeric year,numeric semester){* *numeric p <- 10* *}* This fuction has a void return type represented by none and takes two int type parameters. *numeric method ok(){* *result 1* *}* This fuction has a int return type represented by numeric and takes no parameters.
	Just like Switc_case in C language the switch will take value and case will

Switch-Case	match it.The case block matching the value of switch will be executed and print Case executed and the number of the case block. Syntax: *numeric goal <- 4* *select(&goal){* *pick 3:* *string frnace <- "France wins."* *pick 4:* *string argentina <- "Argentina wins."* *default:* *string draw <- "Match draw."* *}* Pick 4 will be executed and print Case executed : 4
Conditional Statement	In conditional statement if,if..else and if..else if..else are covered.Upon correct evaluation of condition expression desired block will be executed. Syntax: *check (&add > &mul) {* *&a <- &add* *}*

	fallback{ *&a <- &mul* *}* The condition inside check() will be evaluated and according to the result the desired block will be executed.
Expression	Most of the operations are handled through expression like Variable access, number or String return, arithmetic operations, comparison, trigonometric function, math functions, logarithm etc.
Return	What type of value and what value is returned is specified by this. It can be void also.
Loop	For Loop and While Loop are handled here.In for loop it takes initial value,range and steps.It prints macanism happening in each iteration. <u>Syntax:</u> *numeric i<-0,j<-4,k<-1* *iteration(&i : <&j : &k){* *11.11* *}* Output: For iteration 0 OUTPUT: 11.110000 For iteration 1 OUTPUT: 11.110000

	For iteration 2 OUTPUT: 11.110000 For iteration 3 OUTPUT: 11.110000 For while loop it only takes a Int value and execute until a stop condition is not satisfied. Syntax: *conditional_loop(8){* *10* *break(5)* *}* It also prints macanism happening in each iteration.Here break(5) is the stop criteria.
Operators	Used to perform different arithmetic operations. Operator Handled:- ** : Multiplication* *display(2*4) Output: 8* */ : Division* *display(4/2) Output: 2* *+ : Addition* *display(4+2) Output: 6* *- : Subtraction* *display (4-2) Output: 2* *>= : Greater than or equal* *display (11 >= 4.2) Output: 1* *<= : Less than or equal* *display (11 <= 11) Output: 1* *== : Equal to* *display (4 = 4) Output: 1* *!= : Not equal to*

	display (4 != 4) Output: 0 *> : Grater than* *display (4 > 2) Output: 1* *< : Less than* *display (4 < 2) Output: 0*
Built-in Function	Some math functions: log(x) Returns natural logarithm of a number x with base e. *log(10) Output: 2.302585* *pow(2,2) Output: 2.302585* Some Trigonometry : *sin(x)* Returns the sine of an argument (angle in radian). *sin(PI/2) Output: 1* *asin(x)* It takes a single argument (-1 ≤ x ≤ -1), and returns the arc sine (inverse of sin) in radian. *asin(0.5) Output: 0.523599* *cos(x)* Returns the cosine of an argument (angle in radian). *print cos(PI/2); Output: 0* *acos(x)*

	It takes a single argument (-1 ≤ x ≤ -1), and returns the arc cosine (inverse of cosine) in radian. *acos(0.5) Output: 1.047198* *tan(x)* Returns the tangent of an argument (angle in radian). *tan(PI/4) Output: 1* *atan(x)* It takes a single argument, and returns the arc tangent (inverse of tangent) in radian. *atan(0.5) Output: 0.463648*
Symbols	"{" "}" "," ":" "(" ")" [ ]* [\n]* [\t]* .