Khulna University of Engineering & Technology, Khulna

Dept. of Computer Science and Engineering



Course No.: CSE 3212

Course Title: Compiler Design Laboratory

Project Report

Report On : Compiler Design

Submission Date: 21-11-2023

|  |  |
| --- | --- |
| Submitted To | Submitted By |
| Nazia Jahan Khan Chowdhury Assistant Professor  Dept. of CSE, KUET, Khulna  Dipannita Biswas  Lecturer  Dept. of CSE, KUET, Khulna | Mahdi Bhuiyan  Roll: 1907015  Year: 3rd  Term: 2nd  Dept. of CSE, KUET, Khulna |

**Flex:**

flex is a fast lexical analyzer generator.  You specify the scanner you want in the form  of patterns to match and actions to apply for each token. flex is designed for use with C code and generates a scanner written in C and C++.

The scanner is specified using regular expressions for patterns and C or C++ code for the actions. The specifications files are traditionally identified by their .l extension.

To execute the file named ‘a’ with C++ in VS Code we do:

flex a.l

g++ .\lex.yy.c -o app

.\app.exe

**Bison**

Bison, a powerful parser generator, plays a crucial role in the development of compilers and interpreters by automating the creation of parsers for programming languages. Operating in tandem with lexical analyzers like Lex, Bison processes a formal grammar description to define the syntax rules of a language. The grammar, articulated in the context-free grammar (CFG) formalism, outlines the structure of valid programs or expressions. Bison-generated parsers analyze input source code based on these rules, often producing a syntax tree or abstract syntax tree (AST). Bison enables the integration of actions, coded in languages like C, with grammar rules, allowing developers to execute specific tasks during parsing. Recognized for its Yacc compatibility, Bison extends the capabilities of the original tool, providing enhanced features for syntax analysis and parser generation. Through Bison, developers can efficiently create parsers that form the backbone of language processors, facilitating the implementation of compilers and interpreters for a wide range of programming languages.

**Header file:**

#contain<header.h>

Here we can use any header name instead of ‘header’

**Code Start:**

int main()

{

Above portion indicates the start of code section.

**Code Ends:**

}

Above portion indicates the end of code section.

**Data types:**

|  |  |
| --- | --- |
| Type name | In cpp |
| int | integer |
| float | float |

**Comparision Operator:**

|  |  |
| --- | --- |
| Operator Name | Meaning |
| < | Less than |
| > | Greater than |
| <= | Less than equal |
| >= | Greater than equal |

**Mathematical Operator:**

|  |  |
| --- | --- |
| Operator Name | Meaning |
| + | Addition |
| - | Subtruction |
| \* | Multiplication |
| / | Division |

**Variable Declare:**

int a,b=10;

float c,d,e,f=12.2,g;

int m;

int m=1+2+3\*3/4;

So it can accept multiple variables.

**Single Line Comment:**

\*\* indicates single line comment

Eg:

\*\* this is a single line comment

**Loop:**

First type the keyword ‘loop’ then give ‘#’ , then initialization then ‘#’, then condition section, then ‘#’ then increment or decrement then ‘#’ then body part and lastly ‘#’

Eg:

loop # a = 10 # a < 15 # a=a+1 # b=11; #

This is simple loop

loop # a = 10 # a < 15 # a=a+1 #

loop # a = 10 # a < 15 # a=a+1 # b=11; #

#

This is nested loop

Here spaces doesnot matter, except the keyword ‘loop’

**If structure:**

First type the keyword ‘if’ , then give ‘#’ then condition then ‘#’ then statements, lastly ‘#’

Eg:

if # i<j # a=b+c; #

Here spaces doesnot matter, except the keyword ‘if’

if(1<2){ show(x); } else { show(y);

if(1<2){ show(x); } else { show(y); }

Also nested structure it can handle

**User definded Function:**

func int function1 ( 1 ) { int a=10; return 20;}

Here func is keyword , then return type, then function name, then parameters, then body. It must return .

func int function2 ( 1 , 2 ) {

int x=10,y=15;

loop # x # x<15 # x=x+1 # y=10; #

if(1<2){ show(x); }

factorial(5);

return 1;

}

**Built In function:**

prime(14);

factorial(5);

sin ( 1.5 );

cos(0.5);

tan(0.5);

It will be able to calculate this as builtin fuction

|  |  |
| --- | --- |
| Function explain | Structure |
| If Prime | prime(); |
| Find factorial | factorial(); |
| Sine value | sin (); |
| Cos value | cos(); |
| Tan value | tan(); |

**Input file:**

#contain<header.h>

func int function1 ( 1 ) { int a=10; return 20;}

func int function2 ( 1 , 2 ) {

int x=10,y=15;

loop # x # x<15 # x=x+1 # y=10; #

if(1<2){ show(x); }

factorial(5);

return 1;

}

int main()

{

int a,b=10;

int m=1+2+3\*3/4;

function1 ();

function1 (1);

function2 (1,2);

function3();

\*\* comment

loop # a = 10 # a < 15 # a=a+1 # b=11; #

loop # a = 10 # a < 15 # a=a+1 #

loop # a = 10 # a < 15 # a=a+1 # b=11; #

#

prime(14);

factorial(5);

sin ( 1.5 );

cos(0.5);

tan(0.5);

float c;

int d;

c = input(1);

d = input(2.5);

b=11;

show(c);

show(d);

show(c+10);

show(d+10.5);

loop # a = 10 # a < 15 # a=a+1 #

a=1;

if(1<2){ show(x); } else { show(y);

if(1<2){ show(x); } else { show(y); }

}

#

if(1<2){ show(x); }

}

**Output File:**

HEADERFILE included

User defined function started

Variable a is declared

float 20 0

User defined function ended

User defined function started

Variable y is declared

Variable x is declared

LOOP STARTED

value of var is 10

value of var is 11

value of var is 12

value of var is 13

value of var is 14

LOOP ENDED

printing : 10.000000

IF executed

Factorial of 5 = 120

float 1 0

User defined function ended

Main Started.

Variable b is declared

variable a already exists

Variable m is declared

No of parameter do not match

User defined Funtion Called

No of parameter do not match

Function does not exist.

SINGLE\_LINE\_COMMENT

LOOP STARTED

value of var is 10

value of var is 11

value of var is 12

value of var is 13

value of var is 14

LOOP ENDED

LOOP STARTED

LOOP STARTED

value of var is 10

value of var is 11

value of var is 12

value of var is 13

value of var is 14

LOOP ENDED

value of var is 10

value of var is 11

value of var is 12

value of var is 13

value of var is 14

LOOP ENDED

14 is not a prime number.

Factorial of 5 = 120

The value of sin(1.500000) : 0.997495

The value of cos(0.500000) : 0.877583

The value of tan(0.500000) : 0.546302

Variable c is declared

Variable d is declared

User input is at variable c with value 1

User input is at variable d with value 2.500000

printing : 1.000000

printing : 2.000000

printing : 11.000000

printing : 12.500000

LOOP STARTED

printing : 10.000000

printing : 10.000000

printing : 10.000000

printing : 10.000000

IF ELSE executed

IF ELSE executed

value of var is 10

value of var is 11

value of var is 12

value of var is 13

value of var is 14

LOOP ENDED

printing : 10.000000

IF executed

Symbol Table:

Name IsAssigned IntValue FloatValue DataType

a 1 1 - INT

y 1 10 - INT

x 1 10 - INT

b 1 11 - INT

m 1 4 - INT

c 1 - 1.00 FLOAT

d 1 2 - INT

Function Table:

Name No\_of\_Param Return\_Int Return\_Float Return\_Type

function1 1 20 - INT

function2 2 1 - INT

Main ENDED

Parser is finished