

# **KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY**

## Department of Computer Science and Technology

**Title:**  A simple compiler using flex and Bison.

**Course Title:** Compiler Design Laboratory

**Submitted by:**

**Shujoy Kundu**

Roll: ***1907004***

3rd Year 2nd Semester

Department of Computer Science and Engineering, KUET

**Submitted To:**

**Nazia Jahan Khan Chowdhury**

Assistant Professor, Department of Computer Science and Engineering, KUET

**Dipannita Biswas**

Lecturer , Department of Computer Science and Engineering, KUET

**Submission date:**  November 21, 2023

**Objectives:**

* To know how to create different and new semantic and synthetic rules for the compiler.
* To know about shift and reduce policy of a compiler.
* To know about top down and bottom up parser and how they work.
* To build a parser generator using bison.

**Introduction:**

A compiler is a computer program that translates computer code written in one programming language into another language. The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower-level language to create an executable program.

**Flex and Bison:**

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. The tokens generated in this phase is then fed to the parser.

Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1), IELR(1) or canonical LR(1) parser tables. It is used to perform semantic analysis in a compiler. Parsing involves finding the relationship between input tokens. Bison is upward compatible with YACC(Yet Another Compiler Compiler) : all properly-written YACC grammars ought to work with Bison with no change.

Flex and Bison are tools are used together in the development of compilers and interpreters. Flex generates lexical analyzers, breaking down source code into tokens, while Bison generates parsers, analyzing the syntactic structure of the code based on a specified grammar. The two tools communicate through a token stream, with Flex providing tokens to Bison. This collaborative approach helps convert source code into a structured form, such as a parse tree, facilitating further processing for tasks like compilation or interpretation.

%{

#include<bits/stdc++.h>

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<limits.h>

#include<float.h>

using namespace std;

// including the header file to l file

#include "1907004.tab.h"

// include the definiion of the error function

void yyerror(char \*);

%}

integer ("-")?[0-9]+

number ("-")?({integer}([\.][0-9]+)?(e[\+\-]?{integer})|{integer}([\.][0-9]+))

multi\_comments [ ]\*[/]("multi")[/][a-zA-Z0-9!@#\*(){}\_+-,.:\|?><\n\t ]\*[/]("multi")[/]

single\_line\_comment "#"(.\*)

%%

"##end" {return FINISH;}

"import" {

return IMPORT;

}

"read" {

return READ;

}

"print" {

return PRINT;

}

"int" {

return INT;

}

"double" {

return DOUBLE;

}

"boolean" {

return BOOL;

}

"char" {

return CHAR;

}

"string" {

return STRING;

}

"func" {

return FUNC;

}

"void" {

return VOID;

}

"and" {

return AND;

}

"or" {

return OR;

}

("true"|"false") {

string temp = yytext;

yylval.anytype.bval = (temp == "true");

return BOOLD;

}

"for" {

return FOR;

}

"if" {

return IF;

}

"else" {

return ELSE;

}

"while" {

return WHILE;

}

[a-zA-Z\_][a-zA-Z\_0-9]{0,31} {

strcpy(yylval.anytype.sval, yytext);

return IDEN;

}

{integer} {

yylval.anytype.ival = atoi(yytext);

return INTD;

}

{number} {

yylval.anytype.dval = atof(yytext);

return DOUBLED;

}

"+" {

return PLUS;

}

"=" {

return ASSOP;

}

"<" {

return LT;

}

">" {

return GT;

}

"<=" {

return LE;

}

">=" {

return GE;

}

"==" {

return EE;

}

"!=" {

return NE;

}

":" {

return COLON;

}

"-" {

return MINUS;

}

"\*" {

return MUL;

}

"/" {

return DIV;

}

"%" {

return REM;

}

"(" {

return LP;

}

")" {

return RP;

}

"{" {

return LCUR;

}

"}" {

return RCUR;

}

"[" {

return LB;

}

"]" {return RB;}

">>" {return CIN;}

\".\*\" {

strcpy(yylval.anytype.sval, yytext);

return STRINGD;

}

\'.\' {

yylval.anytype.cval = yytext[1];

return CHARD;

}

";" {

return SEMIC;

}

"," {

return COMMA;

}

"." {

return DOT;

}

[ \t\n] {

}

{multi\_comments} {printf("\nMultiple Line comment\n");}

{single\_line\_comment} {printf("\nSingle Line Comment\n");}

. {

printf("Unrecognized character -> %s\n", yytext);

}

%%

**My Parser CFG:**

%{

#include<bits/stdc++.h>

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<limits.h>

#include<float.h>

using namespace std;

// function definition, declaration given below

void yyerror(char \*);

// link the l file with y file

extern int yylex();

map<string,string> var\_type; // map for storing all the variable type and their names

map<string,int> var\_i; // map for stroing the int value

map<string,bool> var\_b; // map for stroing the boolean value

map<string,double> var\_d; // map for stroing the double value

map<string,char> var\_c; // map for stroing the char value

map<string,string> var\_s; // map for storing string value

map<string,int> var\_arr;

map<string,pair<int,int>> arr\_val;

%}

%union {

struct alltype{

char sval[1000];

int ival;

double dval;

int bval;

char cval;

} anytype;

}

%start input

%token<anytype> IDEN IMPORT DOUBLE INT STRING INTD DOUBLED BOOLD DIV MUL PLUS BOOL PRINT READ CIN FINISH

%token<anytype> MINUS REM STRINGD CHAR CHARD FUNC VOID FOR WHILE

%token<anytype> COMMA DOT SEMIC ASSOP LP RP LCUR RCUR COLON GT LT GE LE EE NE IF ELSE AND OR LB RB

%type<anytype> input line library declare double\_declare string\_declare int\_declare OutPut cin terminate

%type<anytype> boolean\_declare char\_declare func\_declare expr term factor number array\_declare

%type<anytype> more\_int more\_double more\_bool more\_char fargs type assign more\_input

%type<anytype> condition

%left OR

%left AND

%left LT GT LE GE EE NE

%left PLUS MINUS

%left DIV MUL

%nonassoc UMINUS

%%

input:

|input line

;

line:

library

|declare

|assign SEMIC

|condition SEMIC

|if

|for

|while

|OutPut

|cin

|terminate

;

terminate:

FINISH {

cout<< "\n\n\t\t\tThe program has terminated" << "\n";

return 0;

}

cin:

READ CIN IDEN more\_input{

string name = $3.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: "<< name <<" variable not declared yet." << "\n";

}

else{

cout << "\nTake input from user from console for " << name <<"\n";

}

}

;

more\_input:

SEMIC

|CIN IDEN more\_input {

string name = $2.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: "<< name <<" variable not declared yet." << "\n";

}

else{

cout << "\nTake input from user from console for " << name <<"\n";

}

}

;

for:

FOR LP f\_first f\_second f\_third RP LCUR input RCUR {

cout << "\nParsing for loop\n";

}

;

while:

WHILE LP condition RP LCUR input RCUR {

cout << "\nParsing while loop\n";

}

;

f\_first:

declare

|assign SEMIC

;

f\_second:

condition SEMIC

;

f\_third:

assign

;

if:

IF LP condition RP LCUR input RCUR elseif {

cout << "\nparsing if block\n";

}

;

elseif:

|else

|ELSE IF LP condition RP LCUR input RCUR elseif {

cout << "\nparsing else if block\n";

}

;

else:

ELSE LCUR input RCUR {

cout << "\nparsing else block\n";

}

;

condition:

expr

|comparison

|logical

;

comparison:

expr EE expr

|expr NE expr

|expr GT expr

|expr LT expr

|expr LE expr

|expr GE expr

;

logical:

|logical\_term

|logical AND logical\_term

|logical OR logical\_term

;

logical\_term:

comparison

|LP logical RP

;

library:

IMPORT IDEN SEMIC {

printf("\nimported library: %s\n", $2.sval);

}

;

OutPut:

PRINT LP IDEN RP SEMIC {

string name = $3.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: "<< name <<" is an invalid variable to print" << "\n";

}

else{

cout<<"\nPrint -> "<<name<<"\n";

}

}

;

declare:

int\_declare

|double\_declare

|string\_declare

|boolean\_declare

|char\_declare

|func\_declare

|array\_declare

;

array\_declare:

INT IDEN LB number RB SEMIC {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of int type has already been declared" << "\n";

}

else {

cout << "\ndeclared : " << name << " int, size -> " << $4.ival << "\n";

var\_type[name] = "INT-arr";

var\_arr[name] = $4.ival; // store the name of the int array

}

}

int\_declare:

INT IDEN ASSOP expr more\_int {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of int type has already been declared" << "\n";

}

else {

cout << "\ndeclared : " << name << " int, value -> " << $4.ival << "\n";

var\_type[name] = "INT";

var\_i[name] = $4.ival; // store the name of the int

}

}

|INT IDEN more\_int {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of int type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": int\n";

var\_type[name] = "INT";

}

}

;

more\_int:

SEMIC

|COMMA IDEN ASSOP expr more\_int {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of int type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": int, value -> " << $4.ival << "\n";

var\_type[name] = "INT";

var\_i[name] = $4.ival;

}

}

|COMMA IDEN more\_int {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of int type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": int\n";

var\_type[name] = "INT";

}

}

;

double\_declare:

DOUBLE IDEN ASSOP expr more\_double {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of double type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": double, value -> " << $4.dval << "\n";

var\_type[name] = "DOUBLE";

var\_d[name] = $4.dval;

}

}

|DOUBLE IDEN more\_double {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of double type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": double\n";

var\_type[name] = "DOUBLE";

}

}

;

more\_double:

SEMIC

|COMMA IDEN ASSOP expr more\_double {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of double type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": double, value -> " << $4.dval << "\n";

var\_type[name] = "DOUBLE";

var\_d[name] = $4.dval;

}

}

|COMMA IDEN more\_double {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of double type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": double\n";

var\_type[name] = "double";

}

}

;

string\_declare:

STRING IDEN ASSOP STRINGD SEMIC {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of string type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ":string, value -> " << $4.sval << "\n";

var\_s[name] = $4.sval;

var\_type[name] = "STRING";

}

}

|STRING IDEN SEMIC {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of string type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ":string\n";

var\_type[name] = "STRING";

}

}

;

boolean\_declare:

BOOL IDEN ASSOP expr more\_bool {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of bool type has already been declared" << "\n";

}

else {

string bval = "false";

if ($4.ival) bval = "true";

cout << "\ndeclared -> " << name << ": bool, value -> " << bval << "\n";

var\_type[name] = "BOOL";

var\_b[name] = $4.bval;

}

}

|BOOL IDEN more\_bool {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of bool type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": bool\n";

var\_type[name] = "BOOL";

}

}

;

more\_bool:

SEMIC

|COMMA IDEN ASSOP expr more\_bool {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of bool type has already been declared" << "\n";

}

else {

string bval = "false";

if ($4.ival) bval = "true";

cout << "\ndeclared -> " << name << ": bool, value -> " << bval << "\n";

var\_type[name] = "BOOL";

var\_b[name] = $4.bval;

}

}

|COMMA IDEN more\_bool {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of bool type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": bool\n";

var\_type[name] = "BOOL";

}

}

;

char\_declare:

CHAR IDEN ASSOP CHARD more\_char {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of char type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": char, value -> '" << $4.cval << "'\n";

var\_type[name] = "CHAR";

var\_c[name] = $4.cval;

}

}

|CHAR IDEN more\_char {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of char type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": char\n";

var\_type[name] = "CHAR";

}

}

;

more\_char:

SEMIC

|COMMA IDEN ASSOP CHARD more\_char {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of char type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": char, value -> '" << $4.cval << "'\n";

var\_type[name] = "CHAR";

var\_c[name] = $4.cval;

}

}

|COMMA IDEN more\_char {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" variable of char type has already been declared" << "\n";

}

else {

cout << "\ndeclared -> " << name << ": char\n";

var\_type[name] = "CHAR";

}

}

;

func\_declare:

FUNC IDEN LP fargs RP COLON type LCUR input RCUR {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: "<< name <<" funciton name has already been declared" << "\n";

}

else {

string ret = $7.sval;

cout << "\nfunction -> " << name << ": return type -> " << ret << "\n";

var\_type[name] = "FUNC";

}

}

;

fargs:

type IDEN {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: redeclared funciton argument -> " << name << ": " << var\_type[name] <<"\n";

}

else {

cout << "\nargument -> " << name << ": " << $1.sval<< "\n";

var\_type[name] = $1.sval;

for (auto &it : var\_type[name]) {

it = toupper(it);

}

}

}

|type IDEN COMMA fargs {

string name = $2.sval;

if (var\_type.find(name) != var\_type.end()) {

cout << "\nError: redeclared function argument -> " << name << ": " << var\_type[name] <<"\n";

}

else {

cout << "\nargument -> " << name << ": " << $1.sval<< "\n";

var\_type[name] = $1.sval;

for (auto &it : var\_type[name]) {

it = toupper(it);

}

}

}

;

assign:

|arr\_assign

|int\_assign

|string\_assign

|char\_assign

|int\_assign COMMA assign

|string\_assign COMMA assign

|char\_assign COMMA assign

;

arr\_assign:

IDEN LB number RB ASSOP number {

string name = $1.sval;

if (var\_type.find(name) == var\_type.end()){

cout<<"\nError: undefined variable -> "<<name <<"\n";

}

else if(var\_type[name] == "INT-arr"){

if (var\_arr[name] > $3.ival && var\_arr[name] >=0 ){

arr\_val[name] = {$3.ival,$6.ival};

cout<<"\nassignment: " << name << "[" << $3.ival << "]" << " -> " << $6.ival << "\n";

}

else{

cout<<"\nError: array size out of bounds -> " << name <<"\n";

}

}

}

;

int\_assign:

IDEN ASSOP expr {

// for int, double

string name = $1.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: undefined variable -> " << name << "\n";

}

else if (var\_type[name] == "INT"){

var\_i[name] = $3.ival;

cout << "\nassignment: " << name << " -> " << $3.ival << "\n";

}

else if (var\_type[name] == "DOUBLE") {

var\_d[name] = $3.dval;

cout << "\nassignment: " << name << " -> " << $3.dval << "\n";

}

else if (var\_type[name] == "BOOL") {

var\_b[name] = $3.bval;

string bval = "false";

if ($3.bval) bval = "true";

cout << "\nassignment: " << name << " -> " << bval << "\n";

}

}

;

string\_assign:

IDEN ASSOP STRINGD {

string name = $1.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: undefined variable -> " << name << "\n";

}

else if (var\_type[name] != "STRING") {

cout << "\nError: invalid value type -> string" << "\n";

}

else {

var\_s[name] = $3.sval;

cout << "\nassignment: " << name << " -> " << $3.sval << "\n";

}

}

;

char\_assign:

IDEN ASSOP CHARD {

string name = $1.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "\nError: undefined variable -> " << name << "\n";

}

else if (var\_type[name] != "CHAR") {

cout << "\nError: invalid value type -> char" << "\n";

}

else {

var\_c[name] = $3.cval;

cout << "\nassignment: " << name << " -> '" << $3.cval << "'\n";

}

}

;

type:

INT {strcpy($$.sval,"int");}

|DOUBLE {strcpy($$.sval,"double");}

|STRING {strcpy($$.sval,"string");}

|CHAR {strcpy($$.sval,"char");}

|BOOL {strcpy($$.sval,"bool");}

|VOID {strcpy($$.sval,"void");}

;

expr:

term {

// only the int and double value

$$.dval = $1.dval;

$$.ival = $1.ival;

}

|MINUS expr %prec UMINUS {

$$.dval = $2.dval;

$$.ival = $2.ival;

}

|expr PLUS term {

$$.dval = $1.dval + $3.dval;

$$.ival = $1.ival + $3.ival;

}

|expr MINUS term {

$$.dval = $1.dval - $3.dval;

$$.ival = $1.ival - $1.ival;

}

;

term:

factor {

$$.dval = $1.dval;

$$.ival = $1.ival;

}

|term MUL factor {

$$.dval = $1.dval \* $3.dval;

$$.ival = $1.ival \* $1.ival;

}

|term DIV factor {

$$.dval = $1.dval / $3.dval;

$$.ival = $1.ival / $1.ival;

}

|term REM factor {

$$.ival = $1.ival % $3.ival;

$$.dval = $$.ival;

}

;

factor:

number {

$$.dval = $1.dval;

$$.ival = $1.ival;

}

|func\_call %prec FCALL

|IDEN {

string name = $1.sval;

if (var\_type.find(name) == var\_type.end()) {

cout << "Error: undeclared variable -> " << name << "\n";

}

else {

if (var\_type[name] == "INT") {

$$.dval = var\_i[name];

$$.ival = var\_i[name];

}

else if (var\_type[name] == "DOUBLE") {

$$.dval = var\_d[name];

$$.ival = var\_d[name];

}

else if (var\_type[name] == "BOOL") {

$$.dval = var\_b[name];

$$.ival = var\_b[name];

}

else if (var\_type[name] == "STRING") {

strcpy($$.sval,var\_s[name].c\_str());

}

else {

$$.cval = var\_c[name];

}

}

}

|LP expr RP {

$$.dval = $2.dval;

$$.ival = $2.ival;

}

;

func\_call:

IDEN LP data RP {

string name = $1.sval;

if (var\_type.find(name) != var\_type.end()) {

if (var\_type[name] != "FUNC") {

cout << "\nError: redeclared function -> " << name << "\n";

}

else {

cout << "\nCalled function name -> " << name << "\n";

}

}

else if (name == "min" || name == "max" || name =="gcd" || name == "input"||name=="output") {

cout << "\nCalled builtin function -> " << name <<"\n";

}

else {

cout << "\nError: undeclared function -> " << name << "\n";

}

}

;

data:

|INTD|DOUBLED|BOOLD|STRINGD|CHARD

|IDEN

|IDEN COMMA data

|INTD COMMA data

|DOUBLED COMMA data

|BOOLD COMMA data

|STRINGD COMMA data

|CHARD COMMA data

;

number:

INTD {

$$.dval = $1.dval;

$$.ival = $1.ival;

}

|DOUBLED {

$$.dval = $1.dval;

$$.ival = $1.dval;

}

|BOOLD {

$$.dval = $1.bval;

$$.ival = $1.bval;

}

;

%%

int main() {

freopen("input.txt","r",stdin);

freopen("output.txt","w",stdout);

yyparse();

return 0;

}

void yyerror(char \*str) {

fprintf(stderr, "error:%s\n", str);

}