***\*program to check validity of an expression***

***prl.l***

%{

#include "prl.tab.h"

%}

%%

[\_a-zA-Z][\_a-zA-Z0-9]\* return ID;

[0-9]+ return NUM;

[ \t] {;}

\n return 0;

. return yytext[0];

%%

***prl.y***

%{

#include<stdio.h>

void yyerror(char \*s);

%}

%token NUM ID

%left '+' '-'

%left '\*' '/'

%%

stmt:exp

;

exp:exp '+' exp

|exp '-' exp

|exp '\*' exp

|exp '/' exp

|'('exp')'

|NUM

|ID

;

%%

int main()

{

printf("\n Enter an expression");

yyparse();

printf("\n Valid Expression");

return 0;

}

void yyerror(char \*s)

{

printf("\n Invalid expression");

}

**For Compilation and execution:**

1. lex prl.l
2. bison –d prl.y
3. cc lex.yy.c prl.tab.c -ll
4. ./a.out

***\*Program to evaluate an expression(1st program)***

***ar.l***

%{

#include "ar.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);

return NUM;}

\n return 0;

. return yytext[0];

%%

***ar.y***

%{

#include<stdio.h>

void yyerror(char \*s);

%}

%token NUM

%left '+' '-'

%%

stmt:exp {printf("%d",$$);};

exp:exp'+'exp {$$=$1+$3;}

|exp'-'exp {$$=$1-$3;}

|NUM {$$=$1;};

%%

void yyerror(char \*s)

{

fprintf(stderr,"%s\n",s);

}

int yywrap()

{

return 1;

}

int main()

{

return yyparse();

}

***\*Evaluation of an expression(2nd program)***

***ar\_cal.l***

%{

#include "ar\_cal.tab.h"

%}

%%

([0-9]+|[0-9]\*\.[0-9]+)+ {yylval.fval=atof(yytext);

return NUM;}

\n return 0;

. return yytext[0];

%%

***ar\_cal.y***

%{

#include<stdio.h>

void yyerror(char \*s);

%}

%union {float fval;}

%token <fval> NUM

%start stmt

%type <fval> stmt e

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%%

stmt:e {printf("%2.2f",$$);}

;

e:e'+'e {$$=$1+$3;}

|e'-'e {$$=$1-$3;}

|e'\*'e {$$=$1\*$3;}

|e'/'e {$$=$1/$3;}

|'-'e %prec UMINUS {$$=-$2;}

|'('e')' {$$=$2;}

|NUM {$$=$1;}

;

%%

OUTPUT:

[priya@localhost ~]$ lex ar\_cal.l

[priya@localhost ~]$ bison -d ar\_cal.y

[priya@localhost ~]$ cc lex.yy.c ar\_cal.tab.c -ll

[priya@localhost ~]$ ./a.out

Enter arithmrtic expression

(3+2)-4

1.00[priya@localhost ~]$ ./a.out

Enter arithmrtic expression

4.5\*2

9.00[priya@localhost ~]$ ./a.out

Enter arithmrtic expression

9.0/0

Inf

***\*Program to convert binary to decimal number***

***bd.l***

%{

#include "bd.tab.h"

extern int yylval;

%}

%%

[0-1]+ {yylval=atoi(yytext);

return NUM;}

\n return 0;

%%

***bd.y***

%{

#include<stdio.h>

int x,p,r,n;

void yyerror(char \*s);

%}

%token NUM

%%

start: NUM{x=$1;p=1;n=0;

while(x!=0)

{

r=x%10;

n=n+r\*p;

p=p\*2;

x=x/10;

}

printf("%d\n",n);

}

%%

main()

{

yyparse();

}

void yyerror(char \*s)

{

printf("Error\n");

}

int yywrap()

{

return 1;

}

***\*Program to convert decimal to binary***

***decbin.l***

%{

#include "decbin.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);

return NUM;}

\n return 0;

%%

***decbin.y***

%{

#include<stdio.h>

int x,p,r,b;

void yyerror(char \*s);

%}

%token NUM

%%

start:NUM {x=$1;p=1;b=0;

while(x!=0)

{

r=x%2;

b=b+(r\*p);

p=p\*10;

x=x/2;

}

printf("%d\n",b);

}

%%

main()

{

yyparse();

}

void yyerror(char \*s)

{

printf("Error\n");

}

int yywrap()

{

return 1;

}

**[Bison Symbols](http://dinosaur.compilertools.net/bison/index.html" \l "SEC91)**

error

A token name reserved for error recovery. This token may be used in grammar rules so as to allow the Bison parser to recognize an error in the grammar without halting the process. In effect, a sentence containing an error may be recognized as valid. On a parse error, the token error becomes the current look-ahead token. Actions corresponding to error are then executed, and the look-ahead token is reset to the token that originally caused the violation. See section [Error Recovery](http://dinosaur.compilertools.net/bison/bison_9.html#SEC81).

YYABORT

Macro to pretend that an unrecoverable syntax error has occurred, by making yyparse return 1 immediately. The error reporting function yyerror is not called. See section [The Parser Function yyparse](http://dinosaur.compilertools.net/bison/bison_7.html#SEC60).

YYACCEPT

Macro to pretend that a complete utterance of the language has been read, by making yyparse return 0 immediately. See section [The Parser Function yyparse](http://dinosaur.compilertools.net/bison/bison_7.html#SEC60).

YYBACKUP

Macro to discard a value from the parser stack and fake a look-ahead token. See section [Special Features for Use in Actions](http://dinosaur.compilertools.net/bison/bison_7.html#SEC67).

YYERROR

Macro to pretend that a syntax error has just been detected: call yyerror and then perform normal error recovery if possible (see section [Error Recovery](http://dinosaur.compilertools.net/bison/bison_9.html#SEC81)), or (if recovery is impossible) makeyyparse return 1. See section [Error Recovery](http://dinosaur.compilertools.net/bison/bison_9.html#SEC81).

YYERROR\_VERBOSE

Macro that you define with #define in the Bison declarations section to request verbose, specific error message strings when yyerror is called.

YYINITDEPTH

Macro for specifying the initial size of the parser stack. See section [Stack Overflow, and How to Avoid It](http://dinosaur.compilertools.net/bison/bison_8.html#SEC80).

YYLEX\_PARAM

Macro for specifying an extra argument (or list of extra arguments) for yyparse to pass to yylex. See section [Calling Conventions for Pure Parsers](http://dinosaur.compilertools.net/bison/bison_7.html#SEC65).

YYLTYPE

Macro for the data type of yylloc; a structure with four members. See section [Textual Positions of Tokens](http://dinosaur.compilertools.net/bison/bison_7.html#SEC64).

yyltype

Default value for YYLTYPE.

YYMAXDEPTH

Macro for specifying the maximum size of the parser stack. See section [Stack Overflow, and How to Avoid It](http://dinosaur.compilertools.net/bison/bison_8.html#SEC80).

YYPARSE\_PARAM

Macro for specifying the name of a parameter that yyparse should accept. See section [Calling Conventions for Pure Parsers](http://dinosaur.compilertools.net/bison/bison_7.html#SEC65).

YYRECOVERING

Macro whose value indicates whether the parser is recovering from a syntax error. See section [Special Features for Use in Actions](http://dinosaur.compilertools.net/bison/bison_7.html#SEC67).

YYSTYPE

Macro for the data type of semantic values; int by default. See section [Data Types of Semantic Values](http://dinosaur.compilertools.net/bison/bison_6.html#SEC44).

yychar

External integer variable that contains the integer value of the current look-ahead token. (In a pure parser, it is a local variable within yyparse.) Error-recovery rule actions may examine this variable. See section [Special Features for Use in Actions](http://dinosaur.compilertools.net/bison/bison_7.html#SEC67).

yyclearin

Macro used in error-recovery rule actions. It clears the previous look-ahead token. See section [Error Recovery](http://dinosaur.compilertools.net/bison/bison_9.html#SEC81).

yydebug

External integer variable set to zero by default. If yydebug is given a nonzero value, the parser will output information on input symbols and parser action. See section [Debugging Your Parser](http://dinosaur.compilertools.net/bison/bison_11.html#SEC86).

yyerrok

Macro to cause parser to recover immediately to its normal mode after a parse error. See section [Error Recovery](http://dinosaur.compilertools.net/bison/bison_9.html#SEC81).

yyerror

User-supplied function to be called by yyparse on error. The function receives one argument, a pointer to a character string containing an error message. See section [The Error Reporting Function yyerror](http://dinosaur.compilertools.net/bison/bison_7.html#SEC66).

yylex

User-supplied lexical analyzer function, called with no arguments to get the next token. See section [The Lexical Analyzer Function yylex](http://dinosaur.compilertools.net/bison/bison_7.html#SEC61).

yylval

External variable in which yylex should place the semantic value associated with a token. (In a pure parser, it is a local variable within yyparse, and its address is passed to yylex.) See section [Semantic Values of Tokens](http://dinosaur.compilertools.net/bison/bison_7.html#SEC63).

yylloc

External variable in which yylex should place the line and column numbers associated with a token. (In a pure parser, it is a local variable within yyparse, and its address is passed to yylex.) You can ignore this variable if you don't use the `@' feature in the grammar actions. See section [Textual Positions of Tokens](http://dinosaur.compilertools.net/bison/bison_7.html#SEC64).

yynerrs

Global variable which Bison increments each time there is a parse error. (In a pure parser, it is a local variable within yyparse.) See section [The Error Reporting Function yyerror](http://dinosaur.compilertools.net/bison/bison_7.html#SEC66).

yyparse

The parser function produced by Bison; call this function to start parsing. See section [The Parser Function yyparse](http://dinosaur.compilertools.net/bison/bison_7.html#SEC60).

%left

Bison declaration to assign left associativity to token(s). See section [Operator Precedence](http://dinosaur.compilertools.net/bison/bison_6.html#SEC51).

%no\_lines

Bison declaration to avoid generating #line directives in the parser file. See section [Bison Declaration Summary](http://dinosaur.compilertools.net/bison/bison_6.html#SEC57).

%nonassoc

Bison declaration to assign nonassociativity to token(s). See section [Operator Precedence](http://dinosaur.compilertools.net/bison/bison_6.html#SEC51).

%prec

Bison declaration to assign a precedence to a specific rule. See section [Context-Dependent Precedence](http://dinosaur.compilertools.net/bison/bison_8.html#SEC76).

%pure\_parser

Bison declaration to request a pure (reentrant) parser. See section [A Pure (Reentrant) Parser](http://dinosaur.compilertools.net/bison/bison_6.html#SEC56).

%raw

Bison declaration to use Bison internal token code numbers in token tables instead of the usual Yacc-compatible token code numbers. See section [Bison Declaration Summary](http://dinosaur.compilertools.net/bison/bison_6.html#SEC57).

%right

Bison declaration to assign right associativity to token(s). See section [Operator Precedence](http://dinosaur.compilertools.net/bison/bison_6.html#SEC51).

%start

Bison declaration to specify the start symbol. See section [The Start-Symbol](http://dinosaur.compilertools.net/bison/bison_6.html#SEC55).

%token

Bison declaration to declare token(s) without specifying precedence. See section [Token Type Names](http://dinosaur.compilertools.net/bison/bison_6.html#SEC50).

%token\_table

Bison declaration to include a token name table in the parser file. See section [Bison Declaration Summary](http://dinosaur.compilertools.net/bison/bison_6.html#SEC57).

%type

Bison declaration to declare nonterminals. See section [Nonterminal Symbols](http://dinosaur.compilertools.net/bison/bison_6.html#SEC53).

%union

Bison declaration to specify several possible data types for semantic values. See section [The Collection of Value Types](http://dinosaur.compilertools.net/bison/bison_6.html#SEC52).

These are the punctuation and delimiters used in Bison input:

`%%'

Delimiter used to separate the grammar rule section from the Bison declarations section or the additional C code section. See section [The Overall Layout of a Bison Grammar](http://dinosaur.compilertools.net/bison/bison_4.html#SEC14).

`%{ %}'

All code listed between `%{' and `%}' is copied directly to the output file uninterpreted. Such code forms the "C declarations" section of the input file. See section [Outline of a Bison Grammar](http://dinosaur.compilertools.net/bison/bison_6.html#SEC35).

`/\*...\*/'

Comment delimiters, as in C.

`:'

Separates a rule's result from its components. See section [Syntax of Grammar Rules](http://dinosaur.compilertools.net/bison/bison_6.html#SEC41).

`;'

Terminates a rule. See section [Syntax of Grammar Rules](http://dinosaur.compilertools.net/bison/bison_6.html#SEC41).

`|'

Separates alternate rules for the same result nonterminal. See section [Syntax of Grammar Rules](http://dinosaur.compilertools.net/bison/bison_6.html#SEC41).