**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester :** 6

**Compiler Design Lab – Assignment 1**

**Lexical Analyser**

**AIM:**

Construct a program to simulate the functionalities of a Lexical Analyser of a compiler using C programming.

**CODE:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char\*kw[]={"int","float","double","char","void","if","else","for","while","do","switch","case","continue","break","void"};

char SP[20]={'{','(',')','}',',',';'};

//char\* fc[]={"main","printf","scanf","sizeof"};

int sing\_flag=0,multi\_flag=0,FC\_flag=0,str\_flag=0,id\_flag=0;

int isletter(char str)

{

if(!((str=='\_') || (str>='A' && str <='Z') || (str>='a' && str <='z')))

return 0;

return 1;

}

void substring(char s[], char sub[], int p, int l)

{

int i=0;

while (p < l)

{

sub[i++] = s[p++];

}

sub[p] = '\0';

}

int is\_digit(char num)

{

if(num>='0' && num<='9')

{

return 1;

}

return 0;

}

int isID(char str[])

{

int i=0;

if(!isletter(str[0]))

return 0;

for(i=1;i<strlen(str);i++)

{

if(isletter(str[i])||is\_digit(str[i]))

continue;

else

return 0;

}

return 1;

}

int is\_digits(char num[])

{

int i;

if(strlen(num)==0)

return 0;

for(i=0;i<strlen(num);i++)

{

if(is\_digit(num[i]))

{

continue;

}

else

return 0;

}

return 1;

}

int isoptfrac(char num[])

{

int c=0,found=0;

for(c=0;c<strlen(num);c++)

{

if(num[c]=='.')

{

break;

found=1;

}

}

if(!found)

{

return 0;

}

if(c>0 && c<strlen(num))

{

char word[20];

substring(num,word,c+1,strlen(num));

if(is\_digits(word))

{

substring(num,word,0,c);

if(is\_digits(word))

return 1;

}

}

return 0;

}

int isoptexp(char num[])

{

int c=0,found=0;

for(c=0;c<strlen(num);c++)

{

if(num[c]=='E')

{

break;

found=1;

}

}

if(!found)

{

return 0;

}

if(c>0 && c<strlen(num))

{

char word[20];

if(num[c+1]=='+'||num[c+1]=='-')

substring(num,word,c+2,strlen(num));

else if(is\_digit(num[c+1]))

substring(num,word,c+1,strlen(num));

else

return 0;

if(is\_digits(word)||isoptfrac(word))

{

substring(num,word,0,c);

if(is\_digits(word)||isoptfrac(word))

return 1;

}

}

return 0;

}

int isSP(char s)

{

int i=0;

for(i=0;i<strlen(SP);i++)

{

if(s==SP[i])

return 1;

}

return 0;

}

int iskeyword(char str[])

{

int i;

int n=sizeof(kw)/sizeof(kw[0]);

for(i=0;i<n;i++)

{

if(!strcmp(str,kw[i]))

return 1;

}

return 0;

}

char\* test\_token(char str[])

{

int i=0;

if(str[strlen(str)-1]=='\n')

str[strlen(str)-1]='\0';

if(multi\_flag==1)

{

if(!strcmp(str,"\*/"))

multi\_flag=0;

return "";

}

if(sing\_flag==1)

return "";

if(FC\_flag==1 || FC\_flag==2)

{

if(FC\_flag==1)

{

if(str[strlen(str)-1]==')')

{

FC\_flag=2;

}

}

else

{

if(str[0]=='(')

{

FC\_flag=1;

}

else if(str[0]==';')

{

FC\_flag=0;

}

}

return "";

}

if(str\_flag==1)

{

if(str[0]=='"')

{

str\_flag=0;

}

return "";

}

if(str[0]=='/')

{

if(str[1]=='\*')

{

multi\_flag=1;

return "MULTI";

}

else if(str[1]=='/')

{

sing\_flag=1;

return "SINGLE";

}

else

return "ARITHOP";

}

else if((str[0]=='+' || str[0]=='\*'|| str[0]=='-'||str[0]=='%') && (str[1]=='\0'))

return "ARITHOP";

else if(isSP(str[0]))

return "SP";

else if(str[0]=='<' || str[0]=='>')

return "RELOP";

else if(str[0]=='=')

{

if(str[1]=='=')

return "RELOP";

else if(str[1]=='\0')

return "ASSIGN";

}

else if(str[0]=='!')

{

if(str[1]=='=')

return "RELOP";

else if(str[1]=='\0')

return "LOGICOP";

}

else if((!strcmp(str,"&&"))||(!strcmp(str,"||")))

return "LOGICOP";

else if(iskeyword(str))

return "KW";

else if(str[0]=='\'')

{

if(str[strlen(str)-1]=='\'')

return "CHARCONST";

}

else if(str[0]=='"')

{

str\_flag=1;

if(str[strlen(str)-1]=='"')

return "STRCONST";

}

else if(is\_digits(str)||isoptfrac(str)||isoptexp(str))

{

return "NUMCONST";

}

else if(isID(str))

{

id\_flag=1;

return "ID";

}

return str;

}

int reset\_flag()

{

FC\_flag=0;

str\_flag=0;

sing\_flag=0;

id\_flag=0;

}

int is\_delimiter(char str)

{

char word[20]={'(',',',')',';','"'};

int i;

for(i=0;i<strlen(word);i++)

{

if(str==word[i])

return 1;

}

return 0;

}

int is\_opr1(char str)

{

char word[20]={'=','+','<','>','!','&','|','-','\*','/','%'};

int i;

for(i=0;i<strlen(word);i++)

{

if(str==word[i])

return 1;

}

return 0;

}

int is\_opr2(char str)

{

char word[20]={'=','&','|','/','\*'};

int i;

for(i=0;i<strlen(word);i++)

{

if(str==word[i])

return 1;

}

return 0;

}

void LA(char stmt[200])

{

int i=0,c=0,j,k=0,ld=0,hold=0;

reset\_flag();

char \* token = strtok(stmt, " ");

char tokens[20][20],word[200]="",word1[20]="";

while( token != NULL )

{

strcpy(tokens[c++],token);

token = strtok(NULL, " ");

}

char sub\_tk[10][20];

while(i<c)

{

//test\_token(tokens[i++]2

for(j=0;j<strlen(tokens[i]);j++)

{

if(is\_delimiter(tokens[i][j]))

{

if(ld!=j)

{

substring(tokens[i],sub\_tk[k],ld,j);

sub\_tk[k++][j-ld]='\0';

}

sub\_tk[k][0]=tokens[i][j];

sub\_tk[k++][1]='\0';

ld=j+1;

}

else if(is\_opr1(tokens[i][j]))

{

if(is\_opr2(tokens[i][j+1]))

{

substring(tokens[i],sub\_tk[k],ld,j);

sub\_tk[k++][j-ld]='\0';

substring(tokens[i],sub\_tk[k],j,j+2);

sub\_tk[k++][2]='\0';

j++;

ld=j+1;

}

else

{

substring(tokens[i],sub\_tk[k],ld,j);

sub\_tk[k++][j-ld]='\0';

sub\_tk[k][0]=tokens[i][j];

sub\_tk[k++][1]='\0';

ld=j+1;

}

}

}

substring(tokens[i],sub\_tk[k],ld,strlen(tokens[i]));

sub\_tk[k++][strlen(tokens[i])-ld]='\0';

for(j=0;j<k;j++)

{

strcpy(word1,test\_token(sub\_tk[j]));

if(hold==1)

{

if(!strcmp(sub\_tk[j],"("))

{

strcpy(word1,"FC");

FC\_flag=1;

hold=0;

}

else

{

strcat(word,"ID ");

hold=0;

}

}

else if(!strcmp(word1,"ID"))

{

hold=1;

}

if(!hold)

{

strcat(word,word1);

if(strcmp(word1,""))

strcat(word," ");

}

if(strcmp(word1,"ID"))

{

id\_flag=0;

}

//printf("%s ",sub\_tk[j]);

}

ld=0;

k=0;

i++;

}

//printf("\n");

if(multi\_flag!=1)

printf("%s\n",word);

else

printf("%s",word);

}

void main()

{

char stmt[128];

FILE \*fp = fopen("test\_file.c", "r");

if(fp == NULL)

{

perror("Unable to open file!");

exit(1);

}

while(fgets(stmt, sizeof(stmt), fp) != NULL)

{

LA(stmt);

strcpy(stmt,"\0");

}

fclose(fp);

}

/\*

MULTI

FC

SP

KW ID ASSIGN NUMCONST SP ID ASSIGN NUMCONST SP

SINGLE

KW SP ID RELOP ID SP

FC

KW

FC

SP

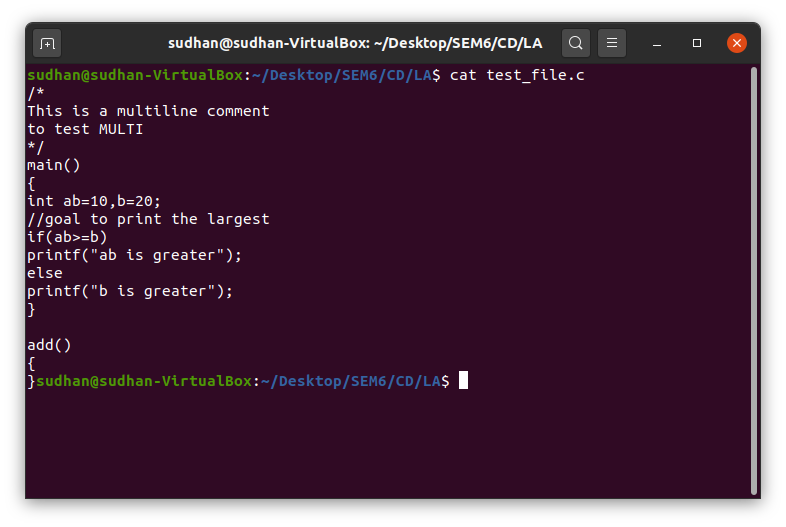
FC

SP

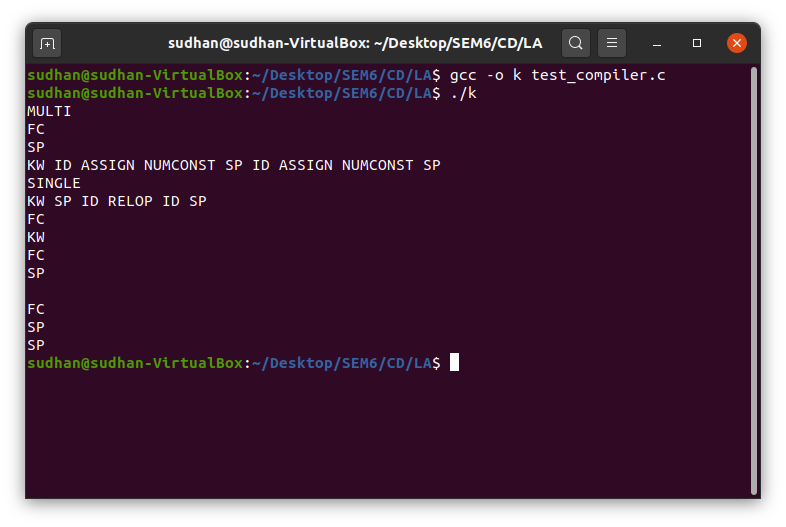
SP

\*/

**INPUT:**

****

**OUTPUT:**

****

**Learning Outcome:**

* I learnt about the role of Lexical analyser in a compiler.
* I understood how the lexical analyser splits the tokens of input statements and organises it.
* I learnt how to tokenize element of a string using inbuilt C functions.
* I also came to know how to differentiate between identifiers and functions.
* Inferred handling of comments , both single line comments and multi-line comments.
* Understood how to differentiate between operators that come in pairs.

**Result:**

Hence the Role of lexical analyser in a compiler is simulated.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester :** 6

**Compiler Design Lab – Assignment 2**

**Lexical Analyser Using LexTool**

**AIM:**

Construct a program to simulate the functionalities of a Lexical Analyser of a compiler using lex Tool.

**CODE:**

%{

#include<stdio.h>

#include<string.h>

int multi\_flag=0,count=0,addr=2000,assign=0,id\_flag=0,dec=0;

char cur\_type[20];

struct symTable{

char sym[20];

int address;

char type[20];

char value[20];

int size;

}tab[10];

void constab(char\*);

void reset\_flag();

void update(char\*);

%}

numberC ([0-9]+)|([0-9]\*(\.)[0-9]+)

charC '(.)'

stringC \"((.)\*)\"

relop ("<"|"<="|"=="|"!="|">"|">=")

arithop ("+"|"-"|"\*"|"/"|"%")

logop ("&&"|"||"|"!")

assign "="

multi\_s \/\\*

multi\_e \\*\/

single "//".\*

SP \(|\)|\{|\}|","|";"

KW if|else|while|do|for|break|continue|switch

data\_type int|float|char|double

function ([ \_ a-z A-Z ])([ \_ a-z A-Z 0-9 ])\*\((.)\*\)

id ([\_a-zA-Z])([\_a-zA-Z0-9])\*

newline "\n"

%%

{numberC} {if(!multi\_flag) printf("NUMCONST ");if(assign==1){update(yytext);reset\_flag();}}

{charC} {if(!multi\_flag) printf("CHARCONST ");if(assign==1){update(yytext);reset\_flag();}}

{stringC} {if(!multi\_flag) printf("STRCONST ");if(assign==1){update(yytext);reset\_flag();}}

{relop} {if(!multi\_flag) printf("RELOP ");reset\_flag();}

{arithop} {if(!multi\_flag) printf("ARITHOP ");reset\_flag();}

{logop} {if(!multi\_flag) printf("LOGOP ");reset\_flag();}

{assign} {if(!multi\_flag) printf("ASSIGN "); if(id\_flag==1) assign=1; }

{multi\_s} {if(!multi\_flag) printf("MULTI\_S ");multi\_flag=1;reset\_flag();}

{multi\_e} {multi\_flag=0;if(!multi\_flag) printf("MULTI\_E "); reset\_flag();}

{single} {if(!multi\_flag) printf("SINGLE ");reset\_flag();}

{SP} {if(!multi\_flag) printf("SP ");reset\_flag();}

{KW} {if(!multi\_flag) printf("KW "); reset\_flag();}

{data\_type} {if(!multi\_flag) printf("KW"); strcpy(cur\_type,yytext); dec=1; }

{function} {if(!multi\_flag) printf("FN ");reset\_flag();}

{newline} {if(!multi\_flag) printf("\n"); dec=0;}

{id} {if(!multi\_flag) printf("ID "); if(dec==1){constab(yytext); id\_flag=1;}}

%%

void main()

{

int i;

FILE \*fp = fopen("test\_file1.c", "r");

if(fp == NULL)

{

perror("Unable to open file!");

exit(1);

}

yyin=fp;

yylex();

printf("\nNAME TYPE SIZE ADDR VALUE\n");

for(i=0;i<count;i++)

{

printf("%4s %6s %4d %4d %s\n",tab[i].sym,tab[i].type,tab[i].size,tab[i].address,tab[i].value);

}

}

void update(char\* val)

{

strcpy(tab[count-1].value,val);

}

void reset\_flag()

{

assign=0;

id\_flag=0;

}

void constab(char\* id)

{

strcpy(tab[count].sym,id);

strcpy(tab[count].type,cur\_type);

tab[count].address=addr;

if(!strcmp(cur\_type,"int"))

{

addr+=2;

tab[count].size = 2;

}

else if(!strcmp(cur\_type,"float"))

{

addr+=4;

tab[count].size =4;

}

else if(!strcmp(cur\_type,"char"))

{

addr+=1;

tab[count].size =1;

}

else if(!strcmp(cur\_type,"double"))

{

addr+=8;

tab[count].size =8;

}

count++;

update("-");

}

int yywrap(void){}

/\*

MULTI\_S

MULTI\_E

FN

SP

KW ID ASSIGN NUMCONST SP ID ASSIGN NUMCONST SP

SINGLE

KW 'SP ID RELOP ID SP

FN SP

KW

FN SP

SP

FN

SP

SP

NAME TYPE SIZE ADDR VALUE

ab int 2 2000 10

b int 2 2002 20

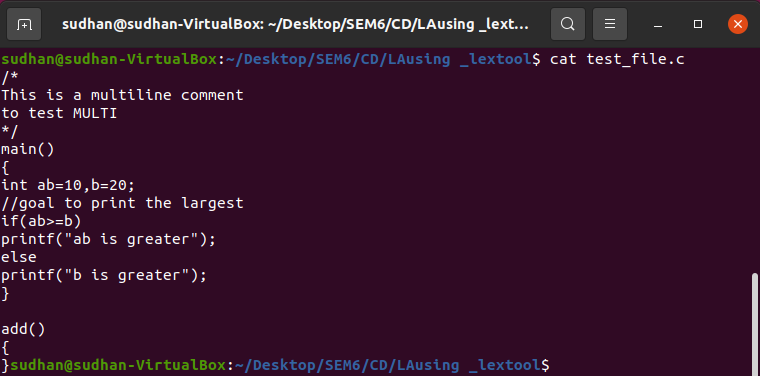
c char 1 2004 'a'

d float 4 2005 9.134

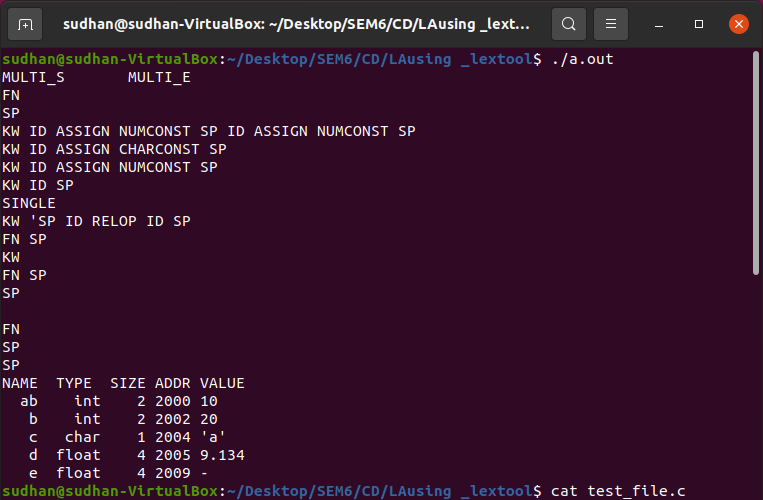
e float 4 2009 -

\*/

**INPUT:**



**OUTPUT:**



**Learning Outcome:**

* I learnt about the role of Lexical analyser in a compiler.
* I understood how the lexical analyser splits the tokens of input statements and organises it.
* I learnt how to tokenize element of a string using lex Tool.
* I also came to know how to how to handle patterns and actions in a lex program.
* Inferred handling of -comments, both single line comments and multi-line comments.
* Understood the use of yywrap() and yylex().

**Result:**

Hence the Role of lexical analyser in a compiler is simulated using a lex tool.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 3**

**Left Recursion Eliminator**

**AIM:**

Construct a program to eliminate the left recursion for a given grammar.

**CODE:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void LReliminator(char prod[]);

void main()

{

char stmt[50];

FILE \*fp = fopen("test\_file.c", "r");

if(fp == NULL)

{

perror("Unable to open file!");

exit(1);

}

while(fgets(stmt, sizeof(stmt), fp) != NULL)

{

LReliminator(stmt);

strcpy(stmt,"\0");

}

fclose(fp);

}

void LReliminator(char prod[])

{

int i,j=0,c=0;

char \* token = strtok(prod,"->"),lhs[20]="",rhs[20]="";

char tokens[20][20],alpha[20];

while( token != NULL )

{

strcpy(tokens[c++],token);

token = strtok(NULL, "->");

}

//printf("%s\n%s\n",tokens[0],tokens[1]);

strcpy(lhs,tokens[0]);

strcpy(rhs,tokens[1]);

for(i=0;i<strlen(lhs);i++)

{

if(lhs[i]==rhs[i])

{

continue;

}

else

{

j=1;

break;

}

}

if(j==1)

{

printf("%s->%s\n",lhs,rhs);

return;

}

for(j=0;rhs[i]!='|';i++,j++)

{

alpha[j]=rhs[i];

}

alpha[i]='\0';

j=-1;

for(i=0;i<strlen(rhs);i++)

{

if(rhs[i]=='|')

{

j=i;

break;

}

}

if(j==-1)

{

printf("LR cant be eliminated\n");

return;

}

c=0;

char \*tok=strtok(rhs,"|");

while( tok != NULL )

{

strcpy(tokens[c++],tok);

tok = strtok(NULL,"|");

}

if(tokens[c-1][strlen(tokens[c-1])-1]=='\n')

{

tokens[c-1][strlen(tokens[c-1])-1]='\0';

}

printf("%s->",lhs);

i=1;

while(i<c)

{

if(i!=c-1)

printf("%s%s'|",tokens[i],lhs);

else

printf("%s%s'",tokens[i],lhs);

i++;

}

printf("\n%s'->(epsilon)|%s%s'\n",lhs,alpha,lhs);

}

/\*

Input File:

E->E+T|T

T->T\*F|F

F->i

Output:

E->TE'

E'->(epsilon)|+TE'

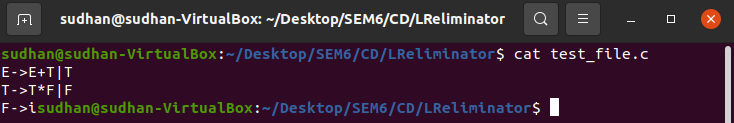
T->FT'

T'->(epsilon)|\*FT'

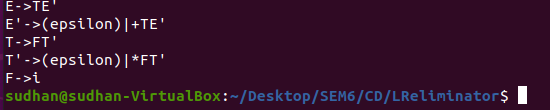
F->i

\*/

**INPUT:**



**OUTPUT:**



**Learning Outcome:**

* I learnt about the problems of the grammar with left recursion.
* I understood how to eliminate Left recursion in a production.
* I learnt how to tokenize element of a string using inbuilt C functions.
* I also came to know how to differentiate the production into alpha and beta .
* Inferred handling of non – left recursion productions.

**Result:**

Hence the Left recursion has been handled and eliminated.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 4**

**Recursive Descent Parser**

**AIM:**

**Write a program in C to construct Recursive Descent Parser for the following grammar which is for arithmetic expression involving + and \*. Check the Grammar for left recursion and convert into suitable for this parser. Write recursive functions for every non-terminal. Call the function for start symbol of the Grammar in main().**

**G: E🡪E+T|T**

**T🡪T\*F | F**

**F🡪i**

**Extend this parser to include division, subtraction and parenthesis operators**

**G: E🡪E+T|E-T|T**

**T🡪T\*F | T/F|F**

**F🡪(E)|i**

**CODE:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char input[100];

int length=0,flag=0,indt=0;

void E();

void T();

void Edash();

void Tdash();

void F();

void indent();

void main()

{

int l;

printf("Enter the Input String:");

scanf("%s",input);

l=strlen(input);

input[l]='$';

input[l+1]='\0';

E();

if(input[length]=='$')

printf("\nThe given string is accepted\n");

else

{

printf("Mismatch at position %d:%c\n",length,input[length]);

printf("String is not accepted\n");

}

}

void E()

{ indent();

indt+=1;

printf(" called E()\n");

T();

Edash();

indt-=1;

}

void Edash()

{

indent();

indt+=1;

printf(" called E'() ");

if(input[length]=='+'||input[length]=='-')

{

if(input[length]=='+')

printf("Matched +\n");

else

printf("Matched -\n");

length++;

T();

Edash();

}

else

printf("\n");

indt-=1;

}

void T()

{

indent();

indt+=1;

printf(" called T()\n");

F();

Tdash();

indt-=1;

}

void Tdash()

{

indent();

indt+=1;

printf(" called T'() ");

if(input[length]=='\*'||input[length]=='/')

{

if(input[length]=='\*')

printf("Matched \*\n");

else

printf("Matched /\n");

length++;

F();

Tdash();

}

else

printf("\n");

indt-=1;

}

void F()

{

indent();

indt+=1;

printf(" called F() ");

indt-=1;

if(input[length]=='i' && input[length+1]=='d')

{

printf("Matched id\n");

length+=2;

}

else if(input[length]=='(')

{

printf("Matched (\n");

length++;

E();

if(input[length]==')')

{

indt+=1;

indent();

indt-=1;

printf("Matched )\n");

length++;

}

else

{

printf("Mismatch at position %d:%c\n",length,input[length]);

printf("String is not accepted\n");

exit(1);

}

}

else

{

printf("Mismatch at position %d:%c\n",length,input[length]);

printf("String is not accepted\n");

exit(1);

}

}

void indent()

{

int i;

for(i=0;i<indt;i++)

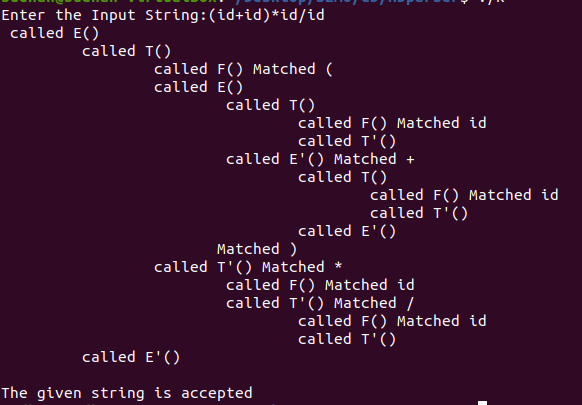
{

printf("\t");

}

}

**Input / Output:**



**Learning Outcome:**

* I learnt about the Recursive Descent Parser.
* I understood how to trace the flow of the recursive Descent parser.
* I learnt how to indent different flows of code.
* I also came to know how and when to throw an error.
* Inferred handling errors in input.

**Result:**

Hence the Recursive Descent Parser has been implemented for the input.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 5**

**Calculator using lex and yacc**

**AIM:**

To implement a calculator using lex and yacc tools.

**Code:**

**LEX:**

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

void yyerror(char\*);

extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

[-+\*/^()=&|%] {return \*yytext;}

"<"|">" {return \*yytext;}

">=" {return GTE;}

"<=" {return LTE;}

"!=" {return NE;}

"==" {return EQ;}

"&&" {return AND;}

"||" {return OR;}

"!" {return NOT;}

"<<" {return LS;}

">>" {return RS;}

[\t] ;

[\n] return 0;

. {yyerror("invalid case");}

%%

int yywrap(void)

{

return 1;

}

**YACC:**

%{

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include"y.tab.h"

int yylex(void);

void yyerror(char \*str);

%}

%token NUMBER OR AND NOT GTE LTE NE EQ LS RS

%right '='

%left OR

%left AND

%left NOT

%left '|'

%left '&'

%left EQ NE

%left '<' '>' GTE LTE

%left LS RS

%left '+' '-' '%'

%left '\*' '/'

%left '^'

%left '(' ')'

%%

Expression: E {

printf("\nResult=%d\n", $$);

return 0;

};

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

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

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

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

|E'%'E {$$=$1%$3;}

|E'^'E {$$=pow($1,$3);}

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

|NOT E {$$=(!$2);}

|E'<'E {$$=$1<$3;}

|E'>'E {$$=$1>$3;}

|E'&'E {$$=$1&$3;}

|E'|'E {$$=$1|$3;}

|E GTE E {$$=$1>=$3;}

|E LTE E {$$=$1<=$3;}

|E EQ E {$$=$1==$3;}

|E NE E {$$=$1!=$3;}

|E AND E {$$=$1&&$3;}

|E OR E {$$=$1||$3;}

|E LS E {$$=$1<<$3;}

|E RS E {$$=$1>>$3;}

| '-' E { $$ = (-($2));}

|NUMBER {$$=$1;}

;

%%

void yyerror(char \*str)

{

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

}

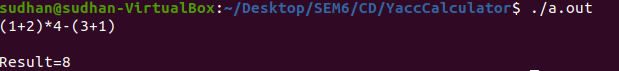
void main()

{

yyparse();

}

**INPUT/OUTPUT:**



**Learning Outcome:**

* I learnt about the Yacc tool and how to combine it with lex to parse a grammar.
* I understood how to write a grammar in yacc.
* I learnt how to mention precedence and associativity.
* I also came to know how to handle error in case of error.
* Inferred handling of multi character tokens by tokens.

**Result:**

Hence the Calculator has been implemented using lex and yacc.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 6**

**Syntax Checker**

**AIM:**

To implement a syntax checking algorithm using lex and yacc.

**CODE:**

**LEX:**

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

void yyerror(char\*);

extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

for {return FOR;}

while {return WHILE;}

if {return IF;}

else {return ELSE;}

[a-zA-Z\_]([a-zA-z\_]|[0-9])\* {return ID;}

[{};()] {return \*yytext;}

[-+\*/^()=&|%] {return \*yytext;}

"<"|">" {return \*yytext;}

">=" {return GTE;}

"<=" {return LTE;}

"!=" {return NE;}

"==" {return EQ;}

"&&" {return AND;}

"||" {return OR;}

"!" {return NOT;}

"<<" {return LS;}

">>" {return RS;}

[\t] ;

[\n] ;

[ ] ;

. {yyerror("invalid case");}

%%

int yywrap(void)

{

return 1;

}

**YACC:**

%{

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include"y.tab.h"

int yylex(void);

void yyerror(char \*str);

%}

%token NUMBER OR AND NOT GTE LTE NE EQ LS RS FOR IF ELSE ID WHILE

%right '='

%left OR

%left AND

%left NOT

%left '|'

%left '&'

%left EQ NE

%left '<' '>' GTE LTE

%left LS RS

%left '+' '-' '%'

%left '\*' '/'

%left '^'

%left '(' ')'

%%

S: ST{printf("\nNo Syntax Error\n");return 0;};

ST: FOR'('SA';'C';'E')'BLOCK

|WHILE'('C')'BLOCK

;

BLOCK:'{'BODY'}'

|BODY

;

BODY:BODY BODY

|SA';'

|IF'('C')'BLOCK ELSE BLOCK

|IF'('C')'BLOCK

|ST

|

;

SA:ID'='E

|E'+''+'

|E'-''-'

;

E:E'+'E

|E'-'E

|E'\*'E

|E'/'E

|E'%'E

|E'^'E

|'('E')'

|E'&'E

|E'|'E

|E LS E

|E RS E

| '-' E

|E'+''+'

|E'-''-'

|NUMBER

|ID

;

C:NOT E

|E'<'E

|E'>'E

|E GTE E

|E LTE E

|E EQ E

|E NE E

|E AND E

|E OR E

%%

void yyerror(char \*str)

{

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

}

void main()

{

yyparse();

}

**INPUT:**

for(i=1;i<10;i++)

{

if(a< b)

x=a;

else

x=b;

}

**OUTPUT:**



**Learning Outcome:**

* I learnt about the syntax checking algorithm.
* I understood how to pass a file as input to a lex and yacc program.
* I learnt how to tokenize elements in order to use them in grammar.
* I also came to know how to throw error in case of wrong syntax.
* Inferred handling of if, for and while syntax.

**Result:**

Hence the Syntax Checker has been implemented and tested.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 7**

**INTERMEDIATE CODE GENERATION**

**AIM:**

To generate intermediate code using lex and yacc.

**CODE:**

**LEX:**

%{

#include<stdio.h>

#include<stdlib.h>

struct data{

int val;

char \*code;

char \*var;

};

#include"y.tab.h"

extern YYSTYPE yylval;

void yyerror(char\*);

%}

%%

[-+]?[0-9]+ {

yylval.val=atoi(yytext);

return NUMBER;

}

[-+]?[0-9]\*[.][0-9]+ {

yylval.val=atoi(yytext);

return NUMBER;

}

for {return FOR;}

while {return WHILE;}

if {return IF;}

then {return THEN;}

else {return ELSE;}

end\_if {return ENDIF;}

integer {return INT;}

real {return REAL;}

char {return CHAR;}

begin {return BEG;}

end {return END;}

var {return VAR;}

['].['] {yylval.val=(int)yytext[1];return NUMBER;}

[a-zA-Z\_]([a-zA-z\_]|[0-9])\* {yylval.info.var=(char\*)malloc(10);strcpy(yylval.info.var,yytext);return ID;}

[{};()] {return \*yytext;}

[-+\*/^()=&|%:] {return \*yytext;}

"<"|">" {return \*yytext;}

">=" {return GTE;}

"<=" {return LTE;}

"!=" {return NE;}

"==" {return EQ;}

"&&" {return AND;}

"||" {return OR;}

"!" {return NOT;}

"<<" {return LS;}

">>" {return RS;}

[\t] ;

[\n] ;

[ ] ;

. {yyerror("invalid case");}

%%

int yywrap(void)

{

return 1;

}

**YACC:**

%{

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

struct SymTable

{

char \*var;

int val;

char \*type;

}s\_table[20];

int yylex(void);

void yyerror(char \*str);

void find(char \*var);

int label=1,t=1,count=0;

void disp();

void find(char \*var)

{

int j=0;

for(j=0;j<count;j++)

{

if(!strcmp(var,s\_table[j].var))

return;

}

char err[100];

sprintf(err,"%s not found\n",var);

yyerror(err);

exit(0);

return;

}

void disp()

{

int j=0;

printf("\tSYMBOL TABLE\n");

printf("Name Type Value\n");

for(j=0;j<count;j++)

{

if(!strcmp(s\_table[j].type,"CHAR"))

{

if(s\_table[j].val==-1)

printf("%-10s %-10s \n",s\_table[j].var,s\_table[j].type);

else

printf("%-10s %-10s %-10c\n",s\_table[j].var,s\_table[j].type,s\_table[j].val);

}

else

printf("%-10s %-10s %-10d\n",s\_table[j].var,s\_table[j].type,s\_table[j].val);

}

}

struct data{

int val;

char \*code;

char \*var;

};

%}

%token NUMBER OR AND NOT GTE LTE NE EQ LS RS FOR IF ELSE ID WHILE INT REAL CHAR BEG END THEN ENDIF VAR

%union{

struct data info;

int val;

char \*code;

}

%type<code> S BLOCK BODY SA;

%type<info> E C ID

%type<val> NUMBER

%right '='

%left OR

%left AND

%left NOT

%left '|'

%left '&'

%left EQ NE

%left '<' '>' GTE LTE

%left LS RS

%left '\*' '/'

%right '+' '-'

%left '%'

%left '^'

%left '(' ')'

%%

S:DECL BEG BLOCK END {printf("BEGIN\n%sEND\n",$3);disp();return 0;};

DECL:DECL DECL

|VAR ID':' INT'='NUMBER';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"INT");s\_table[count++].val=$6;}

|VAR ID':' REAL'='NUMBER';'{s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"REAL");s\_table[count++].val=$6;}

|VAR ID':' REAL';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"REAL");s\_table[count++].val=0;}

|VAR ID':' INT';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"INT");s\_table[count++].val=0;}

|VAR ID':' CHAR'='NUMBER';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"CHAR"); s\_table[count++].val=$6;}

|VAR ID':' CHAR';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"CHAR");s\_table[count++].val=-1;}

|

;

BLOCK:'{'BODY'}' {$$=(char\*)malloc(2000);sprintf($$,"%s",$2);}

|BODY {$$=(char\*)malloc(2000);sprintf($$,"%s",$1);}

;

BODY:BODY BODY {$$=(char\*)malloc(2000);sprintf($$,"%s%s",$1,$2);}

|SA';' {$$=(char\*)malloc(2000);sprintf($$,"%s\n",$1);}

|IF'('C')' THEN BLOCK ELSE BLOCK ENDIF {$$=(char\*)malloc(2000);sprintf($$," if %s goto L%d\n goto L%d\nL%d:\n%s goto L%d\nL%d:\n%sL%d:\n",$3.code,label,label+1,label,$6,label+2,label+1,$8,label+2);label+=3;}

|IF'('C')' THEN BLOCK ENDIF {$$=(char\*)malloc(2000);sprintf($$," if %s goto L%d\n goto L%d\nL%d:\n%sL%d:\n",$3.code,label,label+1,label,$6,label+1);label+=2;}

|BLOCK {$$=(char\*)malloc(2000);sprintf($$,"%s",$1);}

| {$$=(char\*)malloc(2000);strcpy($$,"");}

;

SA:ID'='E {find($1.var);$$=(char\*)malloc(2000); sprintf($$,"%s %s=%s",$3.code,$1.var,$3.var);}

|ID'+''+' {find($1.var);$$=(char\*)malloc(2000);sprintf($$," %s++",$1.var);}

|ID'-''-' {find($1.var);$$=(char\*)malloc(2000);sprintf($$," %s--",$1.var);}

;

E:E'+'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s + %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'-'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s - %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'\*'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s \* %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'/'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s / %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'%'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s mod %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'^'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s ^ %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|'('E')' {$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$2.var);$$.code=(char\*)malloc(300);sprintf($$.code,"%s\n",$2.code);}

|E'&'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s & %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'|'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s | %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E LS E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s << %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E RS E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s >> %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

| '-' E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s=-%s\n",$2.code,$$.var,$2.var);}

|E'+''+'{$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$1.var);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s++\n",$1.code,$1.var);}

|E'-''-'{$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$1.var);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s--\n",$1.code,$1.var);}

|NUMBER {$$.var=(char\*)malloc(10);sprintf($$.var,"%d",$1);$$.code=(char\*)malloc(1);strcpy($$.code,"");}

|ID {find($1.var);$$.var=(char\*)malloc(10);sprintf($$.var,"%s",$1.var);$$.code=(char\*)malloc(1);strcpy($$.code,"");}

;

C:NOT E {$$.code=(char\*)malloc(300);sprintf($$.code,"!%s",$2.var);}

|E'<'E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s < %s",$1.var,$3.var);}

|E'>'E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s > %s",$1.var,$3.var);}

|E GTE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s >= %s",$1.var,$3.var);}

|E LTE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s <= %s",$1.var,$3.var);}

|E EQ E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s == %s",$1.var,$3.var);}

|E NE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s != %s",$1.var,$3.var);}

|E AND E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s && %s",$1.var,$3.var);}

|E OR E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s || %s",$1.var,$3.var);}

%%

void yyerror(char \*str)

{

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

}

void main()

{

yyparse();

}

**INPUT:**

var i:integer=1;

var a:integer=4;

var b:integer=3;

var c:real=6.5;

var d:integer=2;

var x:integer;

var e:char='a';

begin

if (i>0) then

x=a+b\*c/d;

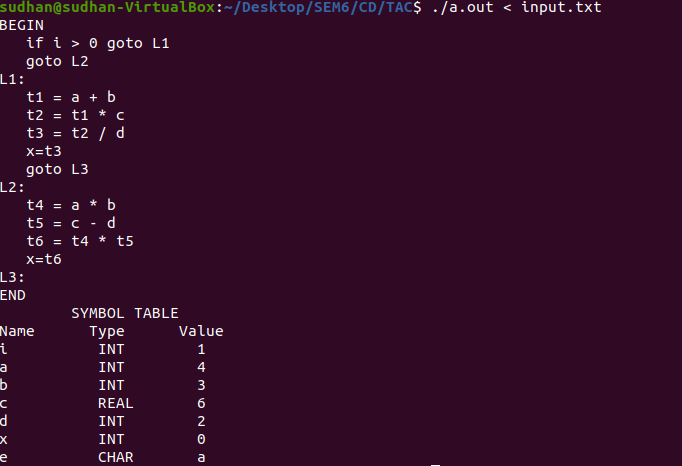
else

x=a\*b\*c-d;

end\_if

end

**OUTPUT:**



**Learning Outcome:**

* I learnt about the Three address code generation and its working.
* I understood how to generate intermediate code for any give grammar.
* I learnt how to pass structure as datatype to lex.
* I also came to know how to set up union of datatypes and use a type for different stacks.
* Inferred handling of errors in productions and also learnt how to throw errors.

**Result:**

Hence the intermediate code generation has been generated using lex and yacc.

**Name:** Sudharshan R

**Registration Number:** 185001173

**Class:** CSE – C **Semester:** 6

**Compiler Design Lab – Assignment 8**

**Code Optimization**

**AIM:**

To perform code Optimization techniques on intermediate code.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void print(char \*input)

{

if(input[0]=='\n'||input[0]=='\0')

{

printf("%c",input[0]);

return;

}

int flag=0;

if(input[3]=='+')

{

if(input[2]=='0')

flag=1;

if(input[4]=='0')

flag=4;

}

if(input[3]=='\*')

{

if(input[2]=='1')

flag=1;

if(input[4]=='1')

flag=4;

if(input[4]=='2')

flag=5;

}

if(input[3]=='/')

{

if(input[4]=='1')

flag=4;

}

if(input[2]=='p'&&input[3]=='o'&&input[4]=='w')

{

if(input[8]=='2')

flag=2;

}

if(input[3]=='-')

{

if(input[2]=='0')

flag=3;

}

if(input[3]=='-')

{

if(input[4]=='0')

flag=4;

}

if(flag==0)

{

printf("%s",input);

}

else if(flag==1)

{

printf("%c=%c\n",input[0],input[4]);

}

else if(flag==2)

{

printf("%c=%c\*%c\n",input[0],input[6],input[6]);

}

else if(flag==3)

{

printf("%c=-%c\n",input[0],input[4]);

}

else if(flag==4)

{

printf("%c=%c\n",input[0],input[2]);

}

else if(flag==5)

{

printf("%c=%c+%c\n",input[0],input[2],input[2]);

}

}

void main(int argc,char \*argv[])

{

char input[1000];

FILE \*fp = fopen("input.txt", "r");

if(fp == NULL)

{

perror("Unable to open file!");

exit(1);

}

int i=0;

while(fgets(input, sizeof(input), fp) != NULL)

{

print(input);

strcpy(input,"\0");

}

printf("\n");

fclose(fp);

}

**INPUT:**

VERSION 1:

a=x+1

b=x+2

c=x+0

a=y\*2

b=y\*4

c=y\*1

VERSION 2:

a=x-1

b=0-x

c=x-0

a=y/1

b=y/2

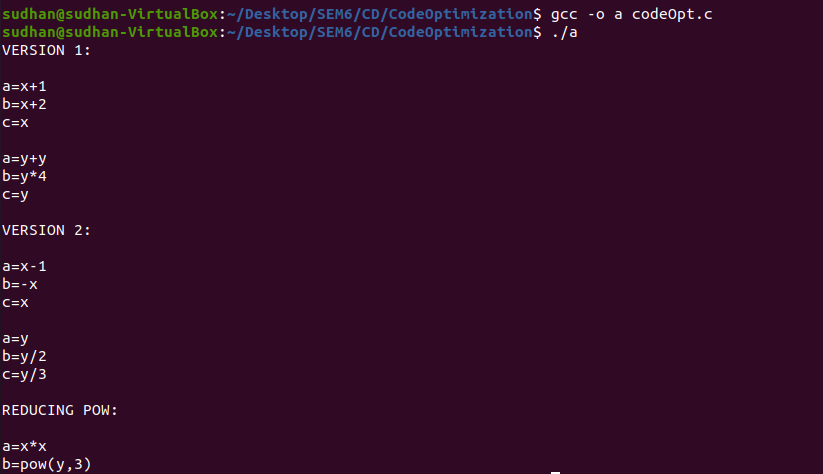
c=y/3

REDUCING POW:

a=pow(x,2)

b=pow(y,3)

**OUTPUT:**



**Learning Outcome:**

* I learnt about the problems of the unoptimized code.
* I understood how to optimize a code provided intermediate code.
* I learnt about Strength reduction techniques in code Optimization.
* I also came to know about pow function reduction.
* Inferred handling + , \* compared to /,-.

**Result:**

Hence code optimization has been done for a sample of intermediate codes.