



**ARUNAI ENGINEERING COLLEGE**

(An Autonomous Institution)

**Velu Nagar, Thiruvannamalai-606 603**  
**[www.arunai.org](http://www.arunai.org)**



**DEPARTMENT OF**  
**COMPUTER SCIENCE & ENGINEERING**

**BACHELOR OF ENGINEERING**

**2024 - 2025**

**FIFTH SEMESTER**

**CS3501 – COMPILER DESIGN LAB**

# ARUNAI ENGINEERING COLLEGE

(An Autonomous Institution)  
TIRUVANNAMALAI – 606 603



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

### CERTIFICATE

Certified that this is a bonafide record of work done by

Name :

University Reg.No :

Semester :

Branch :

Year :

**Staff-in-Charge**

**Head of the Department**

Submitted for the \_\_\_\_\_

Practical Examination held on \_\_\_\_\_

**Internal Examiner**

**External Examiner**

## TABLE OF CONTENTS

EXP /NO.	Date	Experiment Name	Page No	Staff Sign
1		Using the LEX tool, Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, comments, operators etc.). Create a symbol table, while recognizing identifiers		
2		Implement a Lexical Analyzer using LEX Tool		
3		Program to recognize a valid arithmetic expression that Uses operator +, -, * and /.		
4		Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.		
5		Program to recognize a valid control structures syntax of C language (For loop, while loop, if-else, if-else-if, switch-case, etc.).		
6		Implementation of calculator using LEX and YACC		
7		Generate three address code for a simple program using LEX and YACC.		
8		Implement type checking using Lex and Yacc.		
9		Implement simple code optimization techniques (Constant folding, Strength reduction and Algebraic transformation)		
10		Implement back-end of the compiler for which the three address code is given as input and the 8086 assembly language code is produced as output		

## PROGRAM:(1)

//Develop a lexical analyzer to recognize a few patterns in C.

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
void keyword(char str[10])
{
    if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",str)==0||strcmp("float",
str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("printf",str)==0||strcmp("switch",str)==0||
strcmp("case",str)==0)
        printf("\n%s is a keyword",str);
    else
        printf("\n%s is an identifier",str);
}
void main()
{
    FILE *f1,*f2,*f3; char
    c,str[10],st1[10];
    int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
    f1=fopen("input","r"); f2=fopen("identifier","w");
    f3=fopen("specialchar","w");
    while((c=getc(f1))!=EOF)
    {
        if(isdigit(c))
        {
            tokenvalue=c-'0';
            c=getc(f1);
            while(isdigit(c))
            {
                tokenvalue*=10+c-'0';
                c=getc(f1);
            }
            num[i++]=tokenvalue;ungetc(c,f1);
        }
        else
        if(isalpha(c))
        {
            putc(c,f2);
            c=getc(f1);
            while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
            {
                putc(c,f2);
                c=getc(f1);
            }
            putc(' ',f2);
            ungetc(c,f1);
        }
        else
        if(c==' '||c=='\t')
            printf(" ");
    }
```

```
else
if(c=="\n")
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\n the no's in the program are:");
for(j=0;j<i;j++)
printf("\t%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("the keywords and identifier are:");
while((c=getc(f2))!=EOF)
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
fclose(f2);
f3=fopen("specialchar","r"); printf("\n
Special Characters are");
while((c=getc(f3))!=EOF)
printf("\t%c",c);
printf("\n");
fclose(f3);
printf("Total no of lines are:%d",lineno);
}
```

## OUTPUT:

```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ gcc exp2_lexana.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out

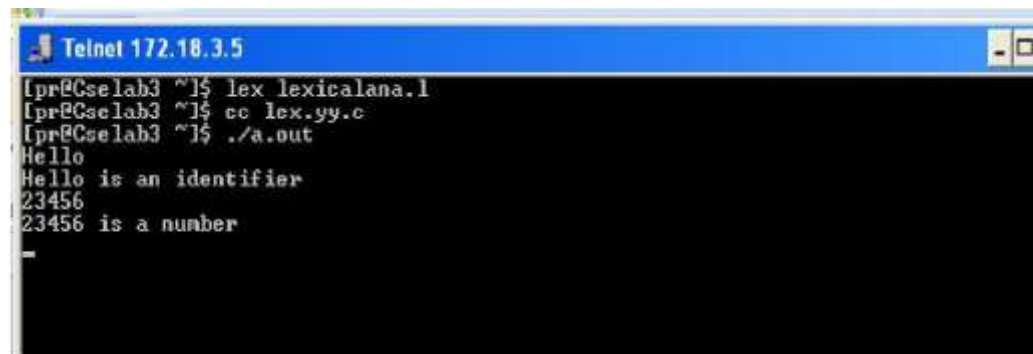
the no's in the program are: 3      2      5
the keywords and identifier are:
int is a keyword
a is an identifier
t1 is an identifier
t2 is an identifier
if is a keyword
printf is a keyword
n is an identifier
else is a keyword
char is a keyword
t3 is an identifier
c is an identifier
Special Characters are {      [      ]      ,      '      ;      (      >
)      (      "      \      "      )      ;      =      ;      }
Total no of lines are:8
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$
```

## PROGRAM 1:(2)

### Program Name: id.1

```
% {  
    #include<stdio.h>  
% }  
  
%%  
if|else|while|int|switch|for    { printf("%s is a keyword",yytext);}  
[a-zA-Z]([a-zA-Z]|[0-9])*      { printf("%s is an identifier",yytext);}  
[0-9]*                          { printf("%s is a number",yytext);}  
%%  
int main()  
{  
    yylex();  
    return 0;  
}  
int yywrap()  
{  
}
```

OUTPUT:



```
Telnet 172.18.3.5
lpr@Cselab3 ~]$ lex lexicalana.l
lpr@Cselab3 ~]$ cc lex.yy.c
lpr@Cselab3 ~]$ ./a.out
Hello
Hello is an identifier
23456
23456 is a number
-
```



## PROGRAM 2:

```
% {
% }
identifier [a-z|A-Z][a-z|A-Z|0-9]*

%%
    #.*                { printf("\n%s is a preprocessor dir",yytext);}
    int                { printf("\n\t%s is a keyword",yytext);}
    { identifier}\(     { printf("\n\nFUNCTION\n\t%s",yytext);}
    \{                 { printf("\nBLOCK BEGINS");}
    \}                 { printf("\nBLOCK ENDS");}
    { identifier}       { printf("\n%s is an IDENTIFIER",yytext);}
    . |\n
%%
```

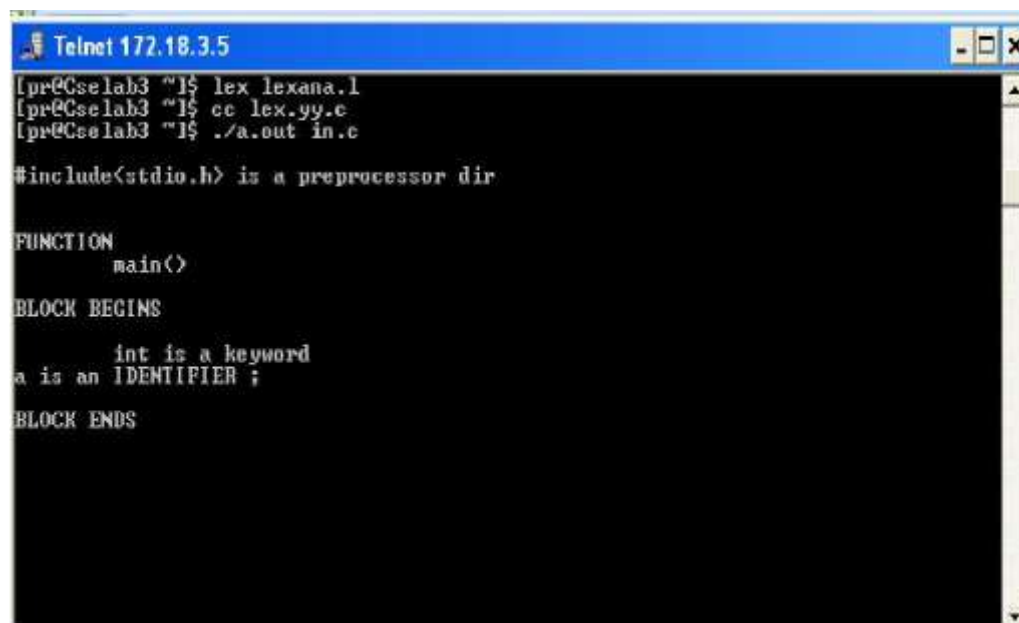
```
int main(int argc,char **argv)
{
    if(argc>1)
    {
        FILE *file;
        file=fopen(argv[1],"r");
        if(!file)
        {
            printf("\n couldnot open %s\n",argv[1]);
            exit(0);
        }
        yyin=file;
    }
    yylex();
    printf("\n\n");
    return 0;
}
```

```
int yywrap()
{
    return 0;
}
```

Input (in.c)

```
#include<stdio.h>
main()
{
    int a ;
}
```

## OUTPUT:



```
Telnet 172.18.3.5
[pr@cse1ab3 ~]$ lex lexana.l
[pr@cse1ab3 ~]$ cc lex.yy.c
[pr@cse1ab3 ~]$ ./a.out in.c

#include<stdio.h> is a preprocessor dir

FUNCTION
    main()

BLOCK BEGINS

    int is a keyword
a is an IDENTIFIER ;

BLOCK ENDS
```

## PROGRAM:(3)

**//art\_expr.l**

```
% {  
    #include<stdio.h>  
    #include "y.tab.h"  
  
% }  
%%  
  
[a-zA-Z][0-9a-zA-Z]* {return ID;}  
[0-9]+ {return DIG;}  
[ \t]+ {;}  
. {return yytext[0];}  
\n {return 0;}  
  
%%  
  
int yywrap()  
{  
    return 1;  
}
```

**//art\_expr.y**

```
% {  
    #include<stdio.h>  
  
% }  
  
%token ID DIG  
%left '+' '-'  
%left '*' '/'  
%right UMINUS  
%%  
  
stmt:expn ;  
expn:expn '+' expn  
    |expn '-' expn  
    |expn '*' expn  
    |expn '/' expn  
    | '-' expn %prec UMINUS  
    | '(' expn ')'  
    | DIG  
    | ID
```

```
    ;  
  
%%  
  
int main()  
{  
    printf("Enter the Expression \n");  
    yyparse();  
    printf("valid Expression \n");  
    return 0;  
}  
  
int yyerror()  
{  
    printf("Invalid Expression");  
    exit(0);  
}
```

## OUTPUT:



```
Telnet 172.18.11.13
"artexp.y" 33L, 372C written
[gomathy@rhel5 ~]$ lex artexp.l
[gomathy@rhel5 ~]$ yacc -d artexp.y
[gomathy@rhel5 ~]$ gcc lex.yy.c y.tab.c
[gomathy@rhel5 ~]$ ./a.out
Enter the Expression
a+b*c-d/e
valid Expression
[gomathy@rhel5 ~]$ ./a.out
Enter the Expression
a=b
Invalid Expression
[gomathy@rhel5 ~]$ _
```

## PROGRAM:(4)

**//valvar.l**

```
% {  
    #include "y.tab.h"  
% }  
%%  
    [a-zA-Z]    {return LET;}  
    [0-9]       {return DIG;}  
    .           {return yytext[0];}  
    \n         {return 0;}  
%%  
int yywrap()  
{  
    return 1;  
}
```

**//valvar.y**

```
% {  
    #include<stdio.h>  
% }  
%token LET DIG  
%%  
    variable:var  
    ;  
    var:var DIG  
    |var LET  
    |LET  
    ;  
%%  
int main()  
{  
    printf("Enter the variable:\n");  
    yyparse();  
    printf("Valid variable \n");  
    return 0;  
}
```

```
}  
int yyerror()  
{  
    printf("Invalid variable \n");  
    exit(0);  
}
```

## OUTPUT:



```
Telnet 172.18.11.13
"valvar.y" 25L, 246C written
[gomathy@rhe15 ~]$ lex valvar.l
[gomathy@rhe15 ~]$ yacc -d valvar.y
[gomathy@rhe15 ~]$ gcc lex.yy.c y.tab.c
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
add
Valid variable
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
add1
Valid variable
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
1add
Invalid variable
[gomathy@rhe15 ~]$
```



## PROGRAM:(5)

**Step 1: Create a new file named "lexer.l" to implement the lexer using Flex (Lex) syntax.**

```
% {
#include "parser.tab.h" // Include the parser header file
% }

%%

[ \t\n]      /* Ignore whitespace */
"for"        { return FOR; }
"while"      { return WHILE; }
"if"         { return IF; }
"else"       { return ELSE; }
"("          { return LPAREN; }
")"          { return RPAREN; }
"{"          { return LBRACE; }
"}"          { return RBRACE; }
";"          { return SEMICOLON; }
[0-9]+       { yylval.str = strdup(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return IDENTIFIER; }
"+"         { return '+'; }
"_"         { return '-'; }
"*"         { return '*'; }
"/"         { return '/'; }

.           { return yytext[0]; } // Any other character is returned as is

%%

int yywrap()
{ return 1;
}
```

**Step 2: Create a new file named "parser.y" to implement the parser using Bison (Yacc) syntax.**

```
% {
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int yylex(); // Declare the lexer function
extern char* yytext;
extern FILE* yyin;
extern int yyparse();
extern int yylineno;

void yyerror(const char* msg) {
    fprintf(stderr, "Error at line %d: %s\n", yylineno, msg);
    exit(1);
}

% }

%union
{ char* str;
}
```

```
;  
  
%%  
  
int main() {  
    yyin = fopen("input_code.c", "r"); // Replace "input_code.c" with the path to your input code file.  
    if (!yyin) {  
        fprintf(stderr, "Error opening input file.\n");  
        return 1;  
    }  
  
    yyparse();  
  
    fclose(yyin);  
    return 0;  
}
```

**Step 3: Compile the lexer and parser files using Flex and Bison respectively.**

```
flex lexer.l  
bison -d parser.y  
gcc lex.yy.c parser.tab.c -o parser -lfl
```

**Step 4: Create a new file named "input\_code.c" and paste the sample input code provided earlier.**

**Step 5: Run the compiled "parser" executable.**

```
bash  
./parser
```

**OUTPUT:**

**If Statement found.  
For loop found.**

## PROGRAM:(6)

**cal.l**

DIGIT [0-9]+

%option noyywrap

% %

```
{DIGIT}      { yylval=atof(yytext);  return NUM;}
\n|.         { return yytext[0];}
```

% %

**cal.y**

```
% {
#include<ctype.h>
#include<stdio.h>
#define YYSTYPE double
% }
%token NUM
%left '+' '-'
%left '*' '/'
%right UMINUS
% %
```

```
Statment:E { printf("Answer: %g \n", $$); }
|Statment '\n'
;
E      : E'+E { $$ = $1 + $3; }
| E'-E { $$=$1-$3; }
| E'*E { $$=$1*$3; }
| E'/E { $$=$1/$3; }
| NUM
;
% %
```

## OUTPUT:

```
"cal2.y" 59L, 1186C written  
[exam01@Cselab3 ~]$ lex cal2.l  
[exam01@Cselab3 ~]$ yacc  
yaccal2.y[exam01@Cselab3 ~]$ cc  
y.tab.c [exam01@Cselab3  
~]$ ./a.out  
Enter the  
expression:2+2Answer:  
4
```

## PROGRAM:(7)

Here's the Lex specification (lexer.l):

```
% {
#include "y.tab.h"
% }

DIGIT    [0-9]
%%
{DIGIT}+ { yylval = atoi(yytext); return NUMBER; }
[ \t\n]  ; // Skip whitespace
.        { return yytext[0]; } // Return other single characters as they are
%%
```

Now, let's define the YACC specification (parser.y) for generating the three-address code:

```
% {
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
void yyerror(const char *msg);

int nextTemporary = 1;

// Helper function to generate a new temporary variable
char* newTemporary() {
    char* temp = (char*)malloc(10);
    sprintf(temp, "t%d", nextTemporary++);
    return temp;
}
% }

%union
{ int
  num;
  char* str;
}

%token <num> NUMBER
%token <str> PLUS MINUS

%left PLUS MINUS

%start program
%%
program: statement
        | program statement
        ;

statement: expr '\n' { printf("%s\n", $1); free($1); }
        ;

expr: NUMBER          { $$ = $1; }
    | expr PLUS expr { char* temp = newTemporary();
                        printf("%s = %s + %s\n", temp, $1, $3);
```

```

        free($1);
        free($3);
        $$ = temp; }
| expr MINUS expr { char* temp = newTemporary();
    printf("%s = %s - %s\n", temp, $1, $3);
    free($1);
    free($3);
    $$ = temp; }

;

%%

void yyerror(const char *msg)
{ fprintf(stderr, "Error: %s\n",
msg);
}

int main()
{ yyparse
();return 0;
}

```

**1. Run Flex to generate the lexer code:**

**flex lexer.l**

This will generate the file **lex.yy.c**.

**2. Run YACC to generate the parser code:**

**yacc -d parser.y**

This will generate the files **y.tab.c** and **y.tab.h**.

**3. Compile the C program:**

`gcc -o parser lex.yy.c y.tab.c`

**4. Create an input file (e.g., input.txt) with the following content:**

`10 + 5 15 - 3`

**5. Run the executable:**

`bashCopy code`

`./parser < input.txt`

**OUTPUT:**

$$t1 = 10 + 5 \quad t2 = t1 - 3$$



## PROGRAM:(8)

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int count=1,i=0,j=0,l=0,findval=0,k=0,kflag=0;
char key[4][12]= {"int","float","char","double"};
char dstr[100][100],estr[100][100];
char token[100],resultvardt[100],arg1dt[100],arg2dt[100];
void entry();
int check(char[]);
int search(char[]);
void typecheck();

struct table
{
char var[10];
char dt[10];
};
struct table tbl[20];

void main()
{
clrscr();
printf("\n IMPLEMENTATION OF TYPE CHECKING \n");

printf("\n DECLARATION \n\n");
do
{
printf("\t");
gets(dstr[i]);
i++;
} while(strcmp(dstr[i-1],"END"));
printf("\n EXPRESSION \n\n");
do
{
printf("\t");
gets(estr[l]);
l++;
} while(strcmp(estr[l-1],"END"));

i=0;
printf("\n SEMANTIC ANALYZER(TYPE CHECKING): \n");
while(strcmp(dstr[i],"END"))
{
entry();
printf("\n");
i++;
}
l=0;

while(strcmp(estr[l],"END"))
{
typecheck();
printf("\n");
}
```

```

        l++;
    }

    printf("\n PRESS ENTER TO EXIT FROM TYPE CHECKING\n");
    getch();
}
void entry()
{
    j=0;
    k=0;
    memset(token,0,sizeof(token));
    while(dstr[i][j]!=' ')
    {
        token[k]=dstr[i][j];
        k++;
        j++;
    }
    kflag=check(token);
    if(kflag==1)
    {
        strcpy(tbl[count].dt,token);
        k=0;
        memset(token,0,strlen(token));
        j++;
        while(dstr[i][j]!=';')
        {
            token[k]=dstr[i][j];
            k++;
            j++;
        }
        findval=search(token);if(findval==0)
        {
            strcpy(tbl[count].var,token);
        }
        else
        {
            printf("The variable %s is already declared",token);
        }

        kflag=0;
        count++;
    }
    else
    {
        printf("Enter valid datatype\n");
    }
}

}

void typecheck()
{
    memset(token,0,strlen(token));
    j=0;
    k=0;
    while(estr[l][j]!='=')

```

```

{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(resultvardt,tbl[findval].dt);findval=0;

}
else
{
    printf("Undefined Variable\n");

}
k=0;
memset(token,0,strlen(token));
j++;
while(((estr[l][j]!='+')&&(estr[l][j]!='-')&&(estr[l][j]!='*')&&(estr[l][j]!='/')))
{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(arg1dt,tbl[findval].dt);
    findval=0;
}
else
{
    printf("Undefined Variable\n");
}
k=0;

memset(token,0,strlen(token));
j++;
while(estr[l][j]!=';')
{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(arg2dt,tbl[findval].dt);
    findval=0;
}
else
{
    printf("Undefined Variable\n");
}

if(!strcmp(arg1dt,arg2dt))
{

```

```

        if(!strcmp(resultvardt,arg1dt))
        {
            printf("\tThere is no type mismatch in the expression %s ",estr[l]);
        }
        else
        {
            printf("\tLvalue and Rvalue should be same\n");
        }
    }
else
{
    printf("\tType Mismatch\n");
}
}

```

```

int search(char variable[])
{
    int i;
    for(i=1;i<=count;i++)
    {
        if(strcmp(tbl[i].var,variable) == 0)
        {
            return i;
        }
    }
    return 0;
}

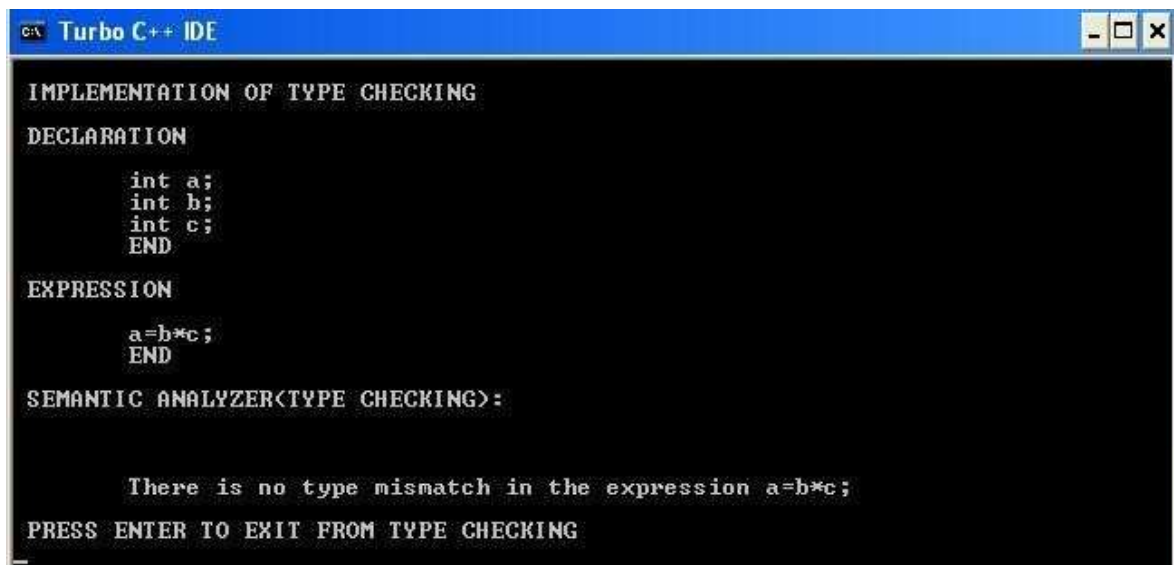
```

```

int check(char t[])
{
    int in;
    for(in=0;in<4;in++)
    {
        if(strcmp(key[in],t)==0)
        {
            return 1;
        }
    }
    return 0;
}

```

## OUTPUT:

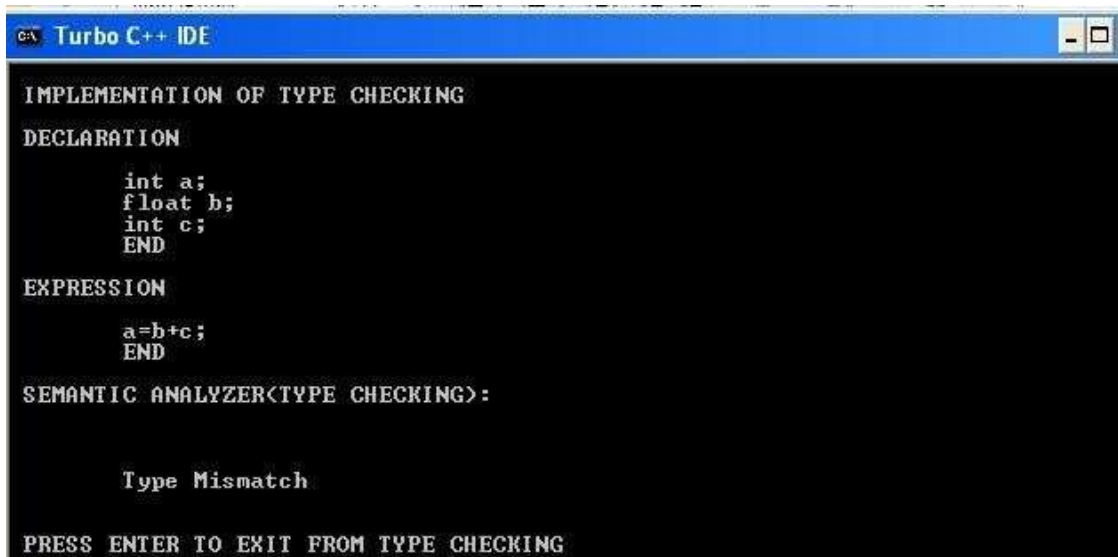


```

Turbo C++ IDE
IMPLEMENTATION OF TYPE CHECKING
DECLARATION
    int a;
    int b;
    int c;
    END
EXPRESSION
    a=b*c;
    END
SEMANTIC ANALYZER<TYPE CHECKING>:

    There is no type mismatch in the expression a=b*c;
PRESS ENTER TO EXIT FROM TYPE CHECKING

```



```

Turbo C++ IDE
IMPLEMENTATION OF TYPE CHECKING
DECLARATION
    int a;
    float b;
    int c;
    END
EXPRESSION
    a=b+c;
    END
SEMANTIC ANALYZER<TYPE CHECKING>:

    Type Mismatch
PRESS ENTER TO EXIT FROM TYPE CHECKING

```

## PROGRAM:(9)

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
    char a[25][25],u,op1='*',op2='+',op3='/',op4='-';
    int p,q,r,l,o,ch,i=1,c,k,j,count=0;
    FILE *fi,*fo;
    // clrscr();
    printf("Enter three address code");
    printf("\nEnter the ctrl-z to complete:\n");
    fi=fopen("infile.txt","w");
    while((c=getchar())!=EOF)
        fputc(c,fi);
    fclose(fi);
    printf("\n Unoptimized input block\n");
    fi=fopen("infile.txt","r");
    while((c=fgetc(fi))!=EOF)
    {
        k=1;
        while(c!=';'&& c!=EOF)
        {
            a[i][k]=c;
            printf("%c",a[i][k]);
            k++;
            c=fgetc(fi);
        }
        printf("\n");
        i++;
    }
    count=i;
    fclose(fi);
    i=1;

    printf("\n Optimized three address code");
    while(i<count)
```

```

{
    if(strcmp(a[i][4],op1)==0&&strcmp(a[i][5],op1)==0)
    {
        printf("\n type 1 reduction in strength");
        if(strcmp(a[i][6],'2')==0)
        {
            for(j=1;j<=4;j++)
                printf("%c",a[i][j]);
            printf("%c",a[i][3]);
        }
    }
    else if(isdigit(a[i][3])&&isdigit(a[i][5]))
    {
        printf("\n type2 constant floding");
        p=a[i][3];
        q=a[i][5];
        if(strcmp(a[i][4],op1)==0)
            r=p*q;
        if(strcmp(a[i][4],op2)==0)
            r=p+q;
        if(strcmp(a[i][4],op3)==0)
            r=p/q;
        if(strcmp(a[i][4],op4)==0)
            r=p-q;
        for(j=1;j<=2;j++)
            printf("%c",a[i][j]);
        printf("%d",r);
        printf("\n");
    }
    else if(strcmp(a[i][5],'0')==0||strcmp(a[i][5],'1')==0)
    {
        cprintf("\n type3 algebraic expression elimation");

        if((strcmp(a[i][4],op1)==0&&strcmp(a[i][5],'1')==0)||(strcmp(a[i][4],op3)==0&&strcmp(a[i][5],'1')==0))
        {
            for(j=1;j<=3;j++)

```

```
        printf("%c",a[i][j]);
    printf("\n");
}

else
    printf("\n sorry cannot optimize\n");
}
else
{
    printf("\n Error input");
}

i++;
}
getch();
}
```



infile.txt

a=d/1; b=2+4; c=s\*\*2;

OUTPUT:

```
Enter three address code
Enter the ctrl-z to complete:
a=d/1;b=2+4;c=s**2;↵

Unoptimized input block
a=d/1
b=2+4
c=s**2

Optimized three address code

type3 algebraic expression elimination: a=d

type2 constant floding: b=6

type 1 reduction in strength: c=s*s
```

## PROGRAM:(10)

```
/* CODE GENERATOR */
#include<stdio.h>
#include<string.h>
int count=0,i=0,l=0;
char str[100][100];
void gen();

void main()
{
    clrscr();
    printf("\n CODE GENERATOR \n");
    printf("\n ENTER THREE ADDRESS CODE \n\n");
    do
    {
        printf("\t");
        gets(str[i]);
        i++;
    } while(strcmp(str[i-1],"QUIT"));

    i=0;
    printf("\n ASSEMBLY LANGUAGE CODE: \n");
    while(strcmp(str[i-1],"QUIT"))
    {
        gen();
        printf("\n");
        i++;
    }

    printf("\n PRESS ENTER TO EXIT FROM CODE GENERATOR\n");
    getch();
}

void gen()
{
    int j;
    printf("\n");
    for(j=strlen(str[i])-1;j>=0;j--)
    {
        char reg='R';
        if(isdigit(str[i][j])||(isalpha(str[i][j]))|| str[i][j]=='+'||str[i][j]=='-'||str[i][j]=='*'||str[i][j]=='/'||str[i][j]==' '||str[i][j]=='&'||str[i][j]=='&'||str[i][j]=='&'||str[i][j]=='&')
        {
            switch(str[i][j])
            {
                case '+':
                    printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
```

```

        printf("\n\t ADD\t%c,%c%d",str[i][j+1],reg,count);
        break;
case '-':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t SUB\t%c,%c%d",str[i][j+1],reg,count);
case '*': break;

        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t MUL\t%c,%c%d",str[i][j+1],reg,count);
case '/': break;

        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
case '|': printf("\n\t DIV\t%c,%c%d",str[i][j+1],reg,count);
        break;

        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t OR\t%c,%c%d",str[i][j+1],reg,count);
        break;
case '&':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t AND\t%c,%c%d",str[i][j+1],reg,count);
        break;
case ':':
        if(str[i][j+1]=='=')
        {
            printf("\n\t MOV\t%c,%c",reg,count,str[i][j-1]);count++;
        }
        else
        {
            printf("\n syntax error...\n");
        }
        break;
default:
        break;
    }
}

else printf("\n Error\n");
}
}

```

**OUTPUT:**

**CODE GENERATOR**

**ENTER THREE ADDRESS**

**CODEA:=B+C**

**D:=E/**

**F**

**QUIT**

**ASSEMBLY LANGUAGE**

**CODE:MOV B,R0**

**ADD C,R0**

**MOV**

**R0,A**

**MOV E,R1**

**DIV F,R1**

**MOV**

**R1,D**

**PRESS ENTER TO EXIT FROM CODE GENERATOR**

