11.YACC program to recognize arithmetic expression

Aim: Write a program using Yacc to test the validity of a simple arithmetic expressionhaving the basic mathematical operators +,-,*,/ and ().

Algorithm:

```
1. Start
2. Define rules in the lex program to generate tokens from the input stream.
    a. Call yylex()
        i.   If input consists of one or more digits from 0 to 9, then pass the
             integer to the parser
        ii.  If '\t', then skip
        iii. if '\n', then return 0.
        iv.  For the remaining, pass the characters to the parser.
    b. Return 1 using yywrap() when input is exhausted
3. In the YACC program,
    a. Initialize the variable ' flag' as 0
    b. Enter an arithmetic expression as input
    c. Call yyparse()
        i.   Define the grammar production and the associated semantic rulesfor
             validating the arithmetic expression.
        ii.  Print the result of the arithmetic expression calculated with the helpof
             semantic rules.
        iii. If error is encountered, yyerror() is called that displays the message
             "Invalid Arithmetic expression' and assigns the flag as 1.
        iv.  If flag is equal to one, then displays the message 'Valid Arithmetic
             Expression'.
4. Stop
```

Program

LEX

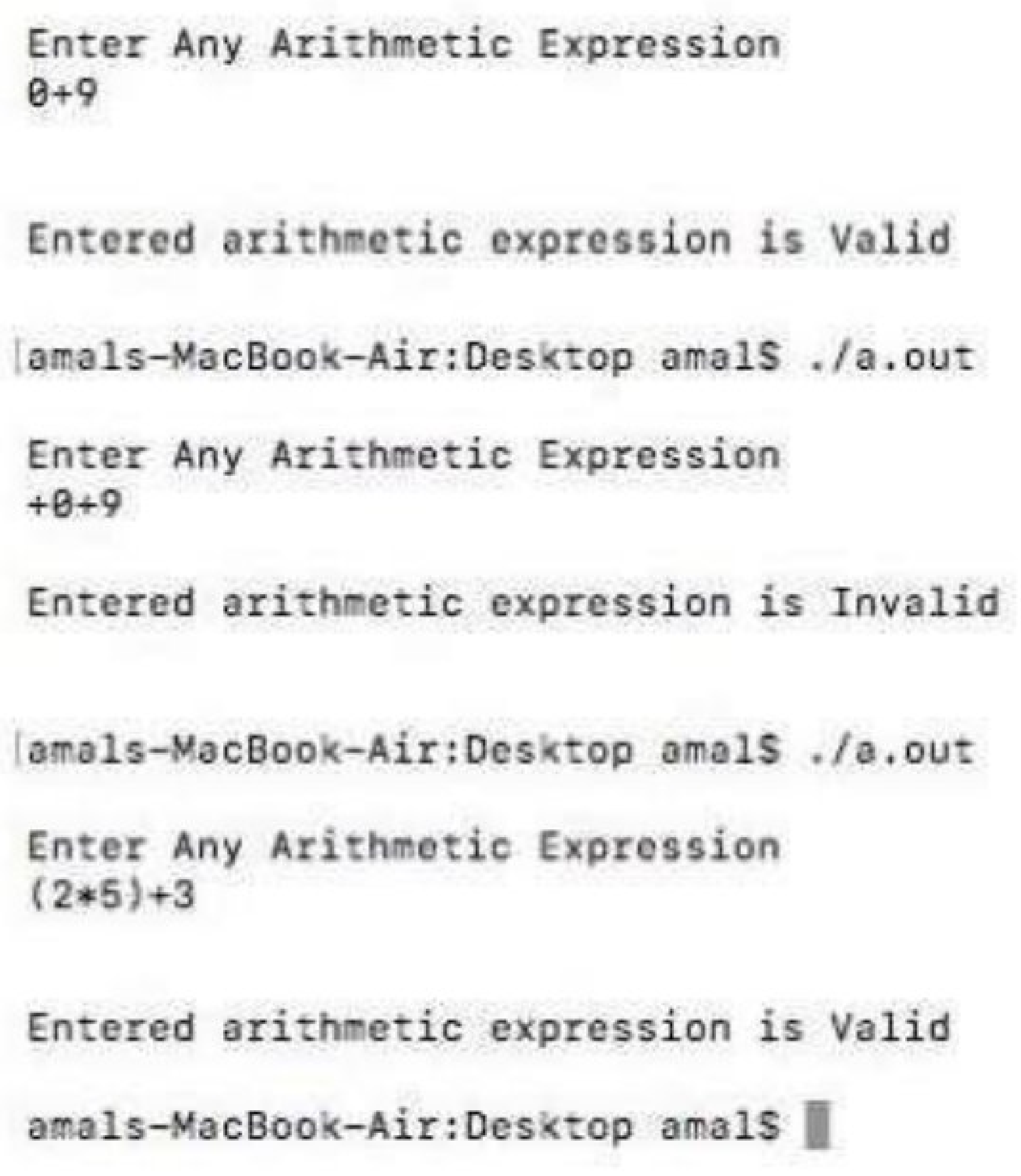
```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h" extern int yylval;
%}
/* Rule Section */
%%
```

```
[0-9]+ {
yylval=atoi(yytext);
return NUMBER;
}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
%}
%token NUMBER
%left '+' '-' %left '*' '/' '%' %left '(' ')'
/* Rule Section */
%%
ArithmeticExpression: E{
return 0;
};
E: E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
//driver code
void main()
{
printf("\nEnter Any Arithmetic Expression\n");
yyparse();
if(flag==0)
printf("\nEntered arithmetic expression is Valid\n\n");
}
int yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
return 0;
}
```

Output:



```
Enter Any Arithmetic Expression
0+9


Entered arithmetic expression is Valid

amals-MacBook-Air:Desktop amal$ ./a.out

Enter Any Arithmetic Expression
+0+9

Entered arithmetic expression is Invalid


amals-MacBook-Air:Desktop amal$ ./a.out

Enter Any Arithmetic Expression
(2*5)+3


Entered arithmetic expression is Valid

amals-MacBook-Air:Desktop amal$
```

Result: Successfully implemented lex program to recognize arithmetic expression.

12.YACC program to check the validity of 'if else statements in C'

Aim: Write a program to check the validity of 'if else statements in C' (using YACC).

Algorithm:

```
1. Start
2. Define rules in the lex program to generate tokens from the input stream.
    a. Call yylex()
        i. if input is "if" return IF
        ii. if input is "else" return ELSE
        iii. If input consists of letter followed by zero or more letter or digits
             return ID
        iv. If input consists of digits return NUM
        v. if input <= return LE or >= return GE
        vi. if input == return EQ or != return NE
        vii. if input || return OR or && return AND
        viii. if '\n', then return 0.
        ix. For the remaining, pass the characters to the parser.
    b. Return 1 using yywrap() when input is exhausted
3. In the YACC program,
    a. Initialize the variable 'valid' as 0
    b. Enter expression as input
    c. Call yyparse()
        i. Define the grammar production and the associated semantic rulesfor
           validating the if else expression.
        ii. If error is encountered, yyerror() is called that displays the message
            "Invalid expression' and assigns the flag as 1.
        iii. If flag is equal to one, then displays the message 'Valid  Expression'.
4. Stop
```

Program

Lex

```
%{
#include "y.tab.h";
extern int yylval;
%}
%%
[ \t]
if return IF;
```

```
else return ELSE;
[a-zA-Z_][a-zA-Z_0-9]* return ID;
[0-9]+(\.[0-9]*)? return NUM;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
"||" return OR;
"&&" return
AND;
. return yytext[0];
\n return 0;
%%
int yywrap(){return 1;}
```

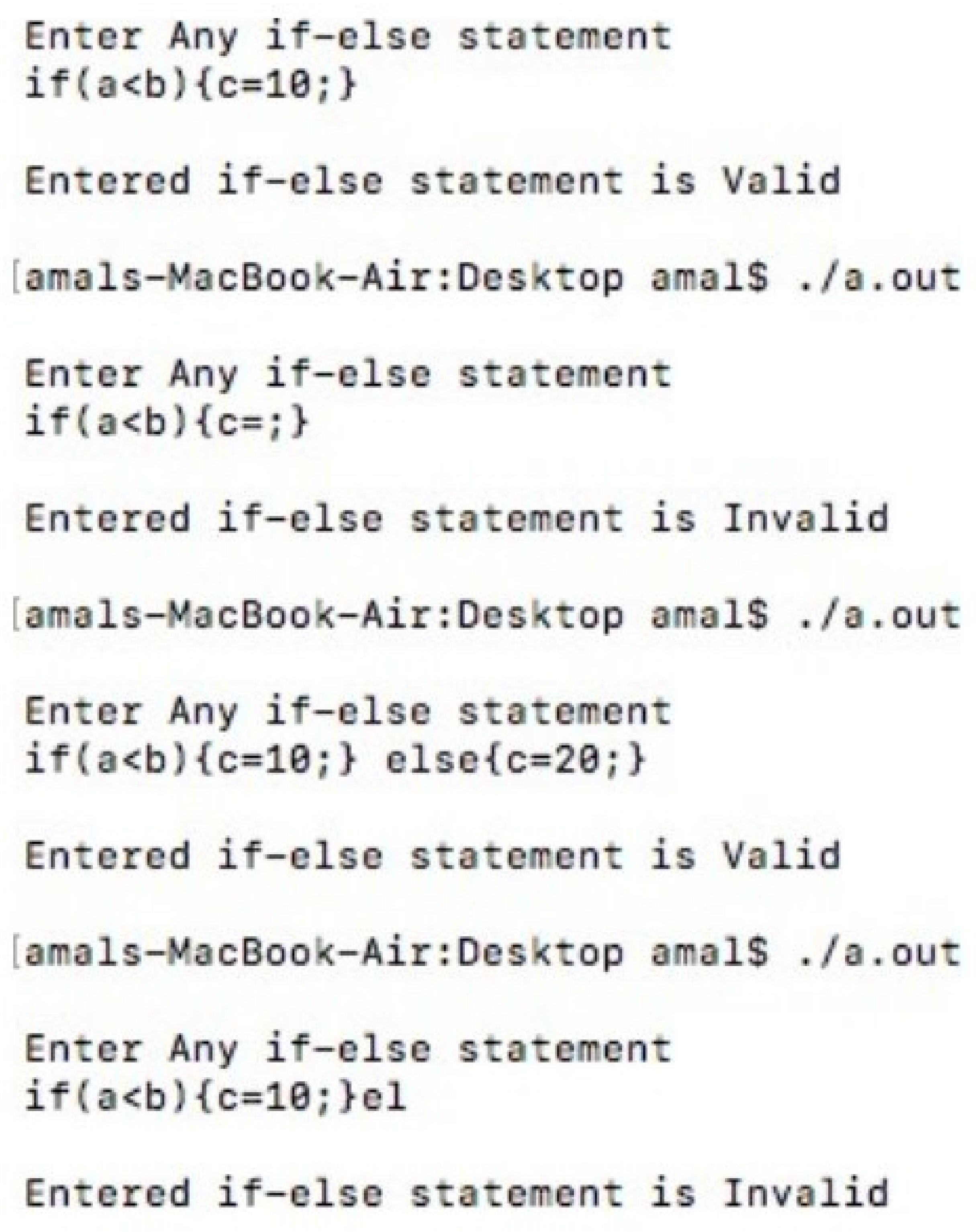
YACC

```
%{
#include<stdio.h>
int valid=0;
%}
%token IF ELSE ID NUM LE GE EQ NE OR AND
%%
start:  statement {valid = 1;};
statement: IF '(' condition ')' '{' ST1';' '}' ELSE '{' ST1';' '}'
        | IF '(' condition ')' '{' ST1';' '}'
        ;
ST1:    statement
        | E
        ;
E:      ID'='E
         | E'+'E
         | E'-'E
         | E'*'E
        | E'/'E
        | E'<'E
        | E'>'E
        | E LE E
        | E GE E
        | E EQ E
        | E NE E
        | E OR E
        | E AND E
        | ID
        | NUM
        ;
condition: E'<'E
        | E'>'E
        | E LE E
        | E GE E
        | E EQ E
        | E NE E
        | E OR E
        | E AND E
        | ID
        | NUM
        ;

%%
//driver code
void main()
{
printf("\nEnter Any if-else statement \n");
yyparse();
if(valid)
printf("\nEntered if-else statement is Valid\n\n");
}

int yyerror()
{
valid=0;
printf("\nEntered if-else statement is Invalid\n\n");return 0;
}
```

Output:



```
 Enter Any if-else statement
 if(a<b){c=10;}

 Entered if-else statement is Valid

[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any if-else statement
 if(a<b){c=;}

 Entered if-else statement is Invalid

[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any if-else statement
 if(a<b){c=10;} else{c=20;}

 Entered if-else statement is Valid

[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any if-else statement
 if(a<b){c=10;}el

 Entered if-else statement is Invalid
```

Result: Successfully implemented program to check the validity of 'if else statements inC'

13.YACC program to check the validity of 'for statements in C'

Aim: Write a program to check the validity of 'for statements in C' (using YACC).

Algorithm:

```
1. Start
2. Define rules in the lex program to generate tokens from the input stream.
    a. Call yylex()
        i.    if input is "for" return FOR
        ii.   If input consists of letter followed by zero or more letter or digits
              return ID
        iii.  If input consists of digits return NUM
        iv.   if input <= return LE or >= return GE
        v.    if input == return EQ or != return NE
        vi.   if input || return OR or && return AND
        vii.  if '\n', then return 0.
        viii. For the remaining, pass the characters to the parser.
    b. Return 1 using yywrap() when input is exhausted
3. In the YACC program,
    a. Initialize the variable 'valid' as 0
    b. Enter expression as input
    c. Call yyparse()
        i.    Define the grammar production and the associated semantic rulesfor
              validating the for expression.
        ii.   If error is encountered, yyerror() is called that displays the message
              "Invalid expression' and assigns the flag as 1.
        iii.  If flag is equal to one, then displays the message 'Valid  Expression'.
4. Stop
```

Program

Lex

```
%{
#include "y.tab.h";
extern int yylval;
%}
alpha [A-Za-z]
digit [0-9]
%%
```

```
[\t]
for             return FOR;
{digit}+   return NUM;
{alpha}({alpha}|{digit})* return
ID;"<="    return LE;
">="        return GE;
"=="        return EQ;
"!="        return NE;
"||"         return OR;
"&&"        return AND;
.              return yytext[0];
\n return 0;
%%
int yywrap(){return 1;}
```

YACC

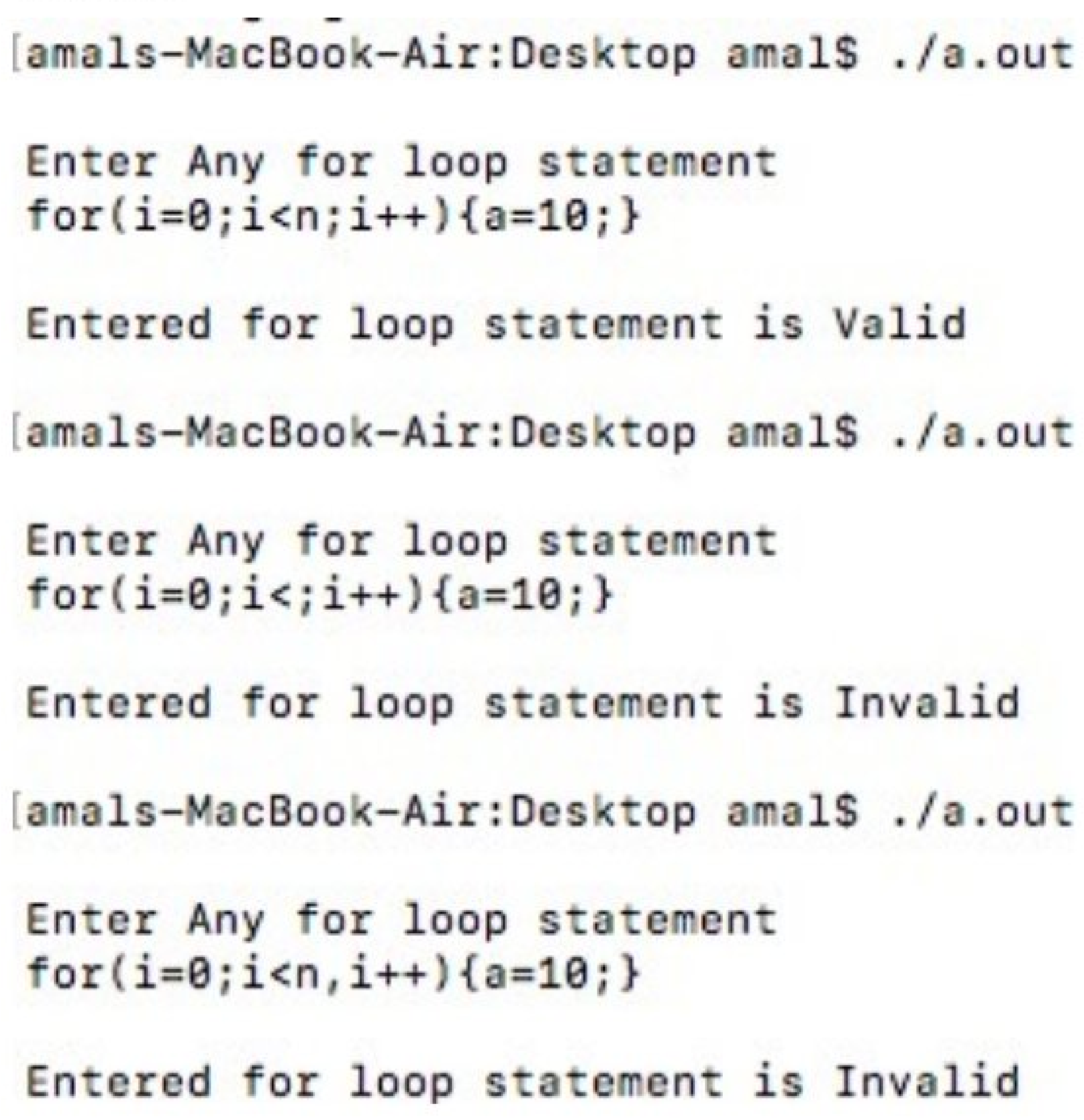
```
%{
#include<stdio.h>
int valid=0;
%}
%token ID NUM FOR LE GE EQ NE OR AND
%right '='
%left OR AND
%left '>' '<' LE GE EQ NE
%left '+' '-'
%left '*' '/'
%right UMINUS
%left '!'
%%
start:  ST {valid = 1;};
ST       : FOR '(' E ';' E2 ';' E ')' DEF
           ;
DEF     : '{' BODY '}'
           | E';'
           | ST
           |
           ;
BODY  : BODY BODY
           | E ';'
           | ST
```

```
         |
         ;

E        : ID '=' E
         | E '+' E
         | E '-' E
         | E '*' E
         | E '/' E
         | E '<' E
         | E '>' E
         | E LE E
         | E GE E
         | E EQ E
         | E NE E
         | E OR E
         | E AND E
         | E '+' '+'
         | E '-' '-'
         | ID
         | NUM
         ;
E2       : E'<'E
         | E'>'E
        | E LE E
        | E GE E
        | E EQ E
        | E NE E
        | E OR E
        | E AND E
        ;
%%
void main()
{
printf("\nEnter Any for loop statement \n");
yyparse();
if(valid)
printf("\nEntered for loop statement is Valid\n\n");
}
```

```
int yyerror()
{
valid=0;
printf("\nEntered for loop statement is Invalid\n\n");return 0;
}
```

Output:



```
[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any for loop statement
 for(i=0;i<n;i++){a=10;}

 Entered for loop statement is Valid

[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any for loop statement
 for(i=0;i<;i++){a=10;}

 Entered for loop statement is Invalid

[amals-MacBook-Air:Desktop amal$ ./a.out

 Enter Any for loop statement
 for(i=0;i<n,i++){a=10;}

 Entered for loop statement is Invalid
```

Result: Successfully implemented program to check the validity of 'for statements in C'