In this folder, I gave the following files: lex.l file (used to generate the lex.yy.c), the lex.yy.c file (used in the parser.y to generate tokens), the parser.y file (used to generate the parser.tab.c file), parser.tab.c file (used to generate the parser)and the parser file (which is final the parser) and the test files.

We first use the Flex to generate the lex.yy.c file. Here is the content:

```
%option noyywrap
%{
    #include<ctype.h>
    #include<string.h>
    #include<stdio.h>
    #include<stdlib.h>
%}
INTEGER ([-+]?[1-9][0-9]*)|[0-9]
DOUBLE [-+]?[0-9]+\.[0-9]*
ID [a-zA-Z][a-zA-Z_0-9]*
SPACE [ \n\t]
COMMENT "/*"[^(*/)|\n]*"*/"|[/][/].*?$
CHAR [\'][^\']*)[\']
%%
{SPACE}      { }
[/][/].*?$ {printf("COMMENT/n");}
"/*"([^("*/")]|\n)*"*/" {printf("COMMENTS/n");}
"if"       {printf("IF  ");return(IF);}
"else"     {printf("ELSE  ");return(ELSE);}
"read"     {printf("READ  ");return(READ);}
"write"    {printf("WRITE  ");return(WRITE);}
"int"      {printf("INT  "); return(BASIC);}
"float"    {printf("FLOAT  "); return(BASIC);}
"break"    {printf("BREAK  ");return(BREAK);}
"do"       {printf("DO  ");return(DO);}
"while"    {printf("WHILE  ");return(WHILE);}
"True"     {printf("TRUE  ");return(TRUE);}
"index"    {printf("INDEX  "); return(INDEX);}
"bool"     {printf("BOOL  "); return(BASIC);}
"char"     {printf("CHAR  "); return(BASIC);}
"real"     {printf("real(double)  ");return(BASIC);}
"False"    {printf("FLASE  "); return(FALSE);}
{ID} {printf("ID ");return(ID);}
{DOUBLE} {printf("DoubleNUM    "); return(DOUBLENUM);}
{INTEGER} { printf("IntegerNUM    ");return(INTEGERNUM); }
{CHAR}   { printf("Char Value  ");return(CHARVALUE);}
"<"        {printf("LT  ");return('<');}
"<="       {printf("LE  ");return(LE);}
"="        {printf("=  ");return('=');}
"=="       {printf("==  ");return(EQ);}
"!="       {printf("!=  ");return(NE);}
"<>"       {printf("!=  ");return(NE);}
">"        {printf(">  ");return('>');}
">="       {printf(">=  ");return(GE);}
"&&"       {printf("and  ");return(AND);}
"||"       {printf("or  ");return(OR);}
"+"        {printf("+  ");return('+');}
"-"        {printf("-  ");return('-');}
"*"        {printf("*  ");return('*');}
```

```
"/"       {printf("/  ");return('/');}
"%"       {printf("%  ");return('%');}
"["       {printf("[  ");return('[');}
"]"       {printf("]  ");return(']');}
"{"       {printf("{  ");return('{');}
"}"       {printf("}  ");return('}');}
"("       {printf("(  ");return('(');}
")"       {printf(")  ");return(')');}
";"       {printf(";  ");return(';');}
","       {printf(",  ");return(',');}
[\r][\n] {}
.       {
    //printf(" illegal character %s\n",  yytext);
    //yyterminate();
}
%%
```

Then we use the parser.y file to generate the parser.tab.c file, here is the content of the .y file:

```
%{
#include<ctype.h>
#include<stdio.h>
#include"lex.yy.c"
%}
%token INTEGERNUM DOUBLENUM CHARVALUE
%token ID
%token IF WHILE DO BREAK TRUE FALSE BASIC ELSE INDEX GE LE NE EQ AND OR READ
WRITE
%%
program : stmts { printf("program-->success\n"); }
        ;
decls :
      | decls decl { printf("decls-->decls decl\n"); }
      ;
decl :
    BASIC IDs ';' { printf("decl-->type id\n"); }
    | BASIC IDs '=' bools ';' { printf("decl-->type id = values\n"); }
    |IDs '=' bools ';' { printf("decl--> id  = values\n"); }

    ;
IDs : ID'['value']' { printf("type-->type id[num]\n"); }
    |  ID { printf("type-->basic"); }
     |IDs','ID { printf("IDs-->ID");}
     ;
bools : bools','booL  { printf("bools-->bool,bools\n"); }
|booL { printf("bools-->bool\n"); }
;
stmts :
    | stmts stmt { printf("stmts-->stmts stmt\n"); }
    | stmts decls {printf("stmts--->stmts decls");}
      ;
stmt : matched_stmt { printf("stmt-->matched_stmt\n");}
     | open_stmt { printf("stmt-->open_stmt\n");}
     ;
open_stmt: IF '(' booL ')' stmt { printf("open_stmt-->if(bool)stmt\n");}
```

```
          | IF '(' booL ')' matched_stmt ELSE open_stmt { printf("open_stmt-->if(bool) matched_stmt
else open_stmt\n");}
        ;
matched_stmt: IF '(' booL ')' matched_stmt ELSE matched_stmt { printf("matched_stmt-->if(bool)
matched_stmt else matched_stmt\n");}
            | other { printf("matched_stmt-->other\n");}
          ;
other:
    loc '=' booL ';' { printf("stmt-->loc=bool;\n"); }
    | WHILE '(' booL ')' stmt { printf("stmt-->while(bool)stmt\n"); }
     | DO stmt WHILE '(' booL ')' ';' { printf("stmt-->do stmt while(bool);\n"); }
     | BREAK ';' { printf("stmt-->break;\n"); }
     | block { printf("stmt-->block\n"); }
     | WRITE '(' value ')';' { printf("stmt-->write\n"); }
     | READ '(' value ')';{ printf("stmt-->read\n"); }
     ;
value : INTEGERNUM { printf("value-->integer\n"); }
     | DOUBLENUM { printf("value-->double\n"); }
     | CHARVALUE { printf("value-->char\n"); }
     | IDs { printf("value-->IDs\n"); }
     | expr { printf("vlaue--> expr"); }
block : '{' decls stmts '}' { printf("block-->{decls stmts}\n"); }
;
loc : loc '[' booL ']' { printf("loc-->loc[bool]\n"); }
|IDs { printf("loc-->IDs\n"); }
;
booL : booL OR join { printf("bool-->bool||join\n"); }
     | join { printf("bool-->join\n"); }
     ;
join : join AND equality { printf("join-->join&&equality\n"); }
     | equality { printf("join-->equality\n"); }
     ;
equality : equality EQ rel { printf("equality-->equality==rel\n"); }
        | equality NE rel { printf("equality-->equality!=rel\n"); }
            | rel { printf("equality-->rel\n"); }
            ;
rel : expr '<' expr { printf("rel-->expr<expr\n"); }
    | expr LE expr { printf("rel-->expr<=expr\n"); }
    | expr GE expr { printf("rel-->expr>=expr\n"); }
    | expr '>' expr { printf("rel-->expr>expr\n"); }
    | expr { printf("rel-->expr\n"); }
    ;
expr : expr '+' term { printf("expr-->expr+term\n"); }
     | expr '-' term { printf("expr-->expr-term\n"); }
     | term { printf("expr-->term\n"); }
     ;
term : term '*' unary { printf("term-->term*unary\n"); }
     | term '/' unary { printf("term-->term/unary\n"); }
    | term '%' unary { printf("term-->term/unary\n"); }
     | unary { printf("term-->unary\n"); }
     ;
unary : '!' unary { printf("unary-->!unary\n"); }
      | '-' unary { printf("unary-->-unary\n"); }
      | factor { printf("unary-->factor\n"); }
      ;
factor : '(' booL ')' { printf("factor-->(bool)\n"); }
```

```
    | loc { printf("factor-->loc\n"); }
      | INTEGERNUM { printf("factor-->num\n"); }
      | DOUBLENUM {printf("factor-->num\n");}
      | TRUE { printf("factor-->true\n"); }
      | CHARVALUE { printf("factor-->char\n"); }
      | FALSE { printf("factor-->false\n"); }
   ;
%%

main(int argc, char** argv)
{
    if(argc > 1)
        yyin = fopen(argv[1], "r");
    else
        yyin = stdin;
//    printf("oh");
    yyparse();
//    return 0;
}
yyerror (char *s){

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

}
```

Then we use the gcc to generate the parser:

```
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$ flex lex.l
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$ bison parser.y
parser.y: conflicts: 24 shift/reduce, 9 reduce/reduce
```

```
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$ gcc -o parser parser.tab.c
In file included from parser.y:4:
lex.l:51:10: warning: incomplete format specifier [-Wformat]
{printf("%  ");return('%');}
         ^~~
parser.tab.c:1812:7: warning: implicit declaration of function 'yyerror' is
      invalid in C99 [-Wimplicit-function-declaration]
      yyerror (YY_("syntax error"));
      ^
parser.y:99:1: warning: type specifier missing, defaults to 'int'
      [-Wimplicit-int]
main(int argc, char** argv)
^
parser.y:109:1: warning: type specifier missing, defaults to 'int'
      [-Wimplicit-int]
yyerror (char *s){
^
parser.y:113:1: warning: control reaches end of non-void function
      [-Wreturn-type]
}
^
5 warnings generated.
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$
```

Then we test the test file, before we test, I change the following code in the origin test file

1. delete all the “*” inside the comment
2. make 2+3 become 2 + 3

3.delete the last comment of all the files

4.in test2.txt change

int aa,ab = 23 , (-4);

into

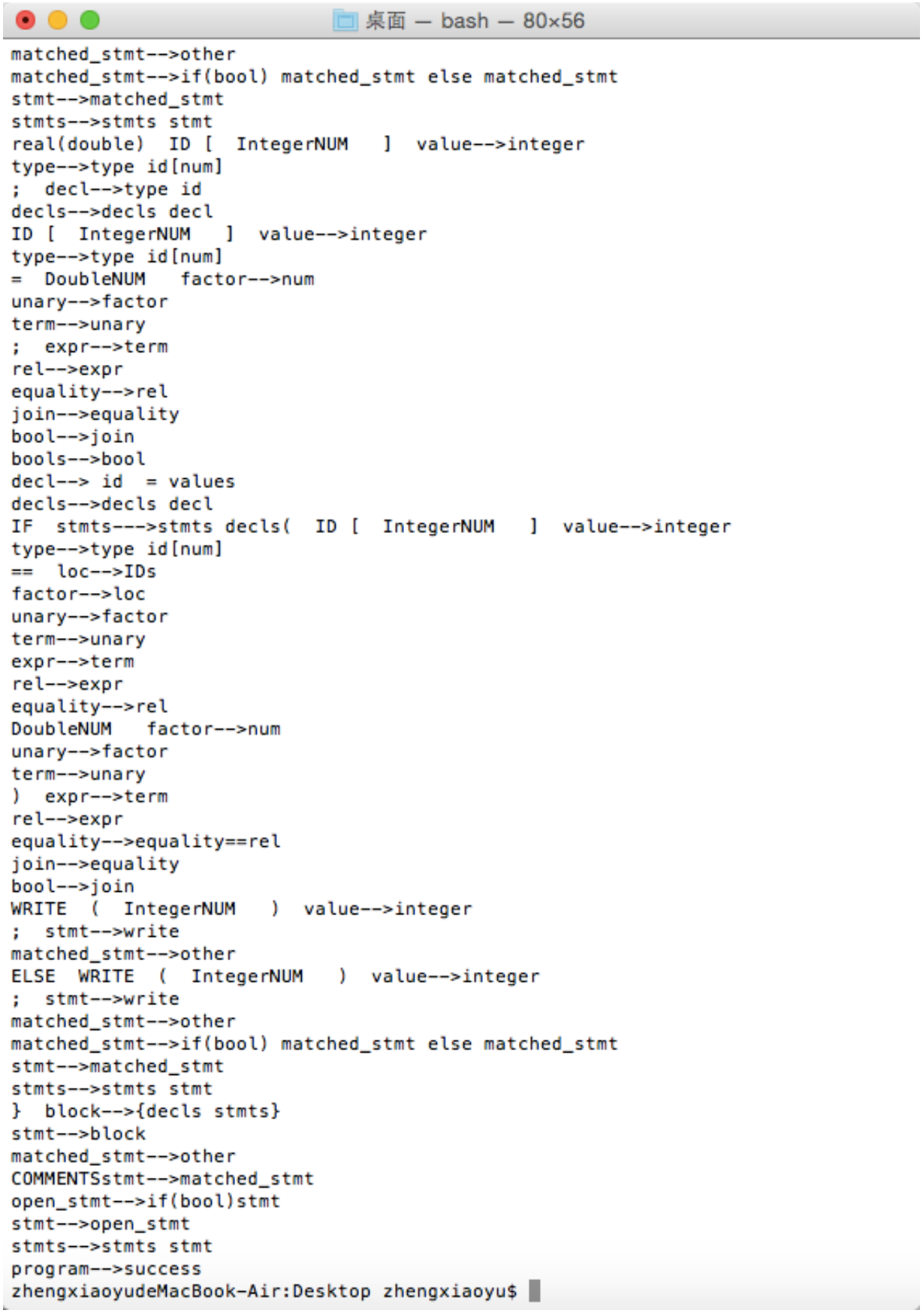
int aa = 23 ;

int ab = (-4);

Then we start to test all those files:

```
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$ ./parser test1.txt
COMMENTSCOMMENTINT  ID ;  type-->basicdecl-->type id
decls-->decls decl
real(double)  ID ;  type-->basicdecl-->type id
decls-->decls decl
real(double)  ID ,  type-->basicID IDs-->ID=  IntegerNUM   factor-->num
unary-->factor
term-->unary
,  expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool
ID ;  type-->basicloc-->IDs
factor-->loc
unary-->factor
term-->unary
expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool,bools
decl-->type id = values
decls-->decls decl
COMMENTINT  ID =  type-->basicIntegerNUM   factor-->num
unary-->factor
term-->unary
;  expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool
decl-->type id = values
decls-->decls decl
real(double)  ID [  IntegerNUM   ]  value-->integer
type-->type id[num]
;  decl-->type id
decls-->decls decl
ID [  IntegerNUM   ]  value-->integer
type-->type id[num]
=  IntegerNUM   factor-->num
unary-->factor
term-->unary
;  expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool
decl--> id  = values
decls-->decls decl
```

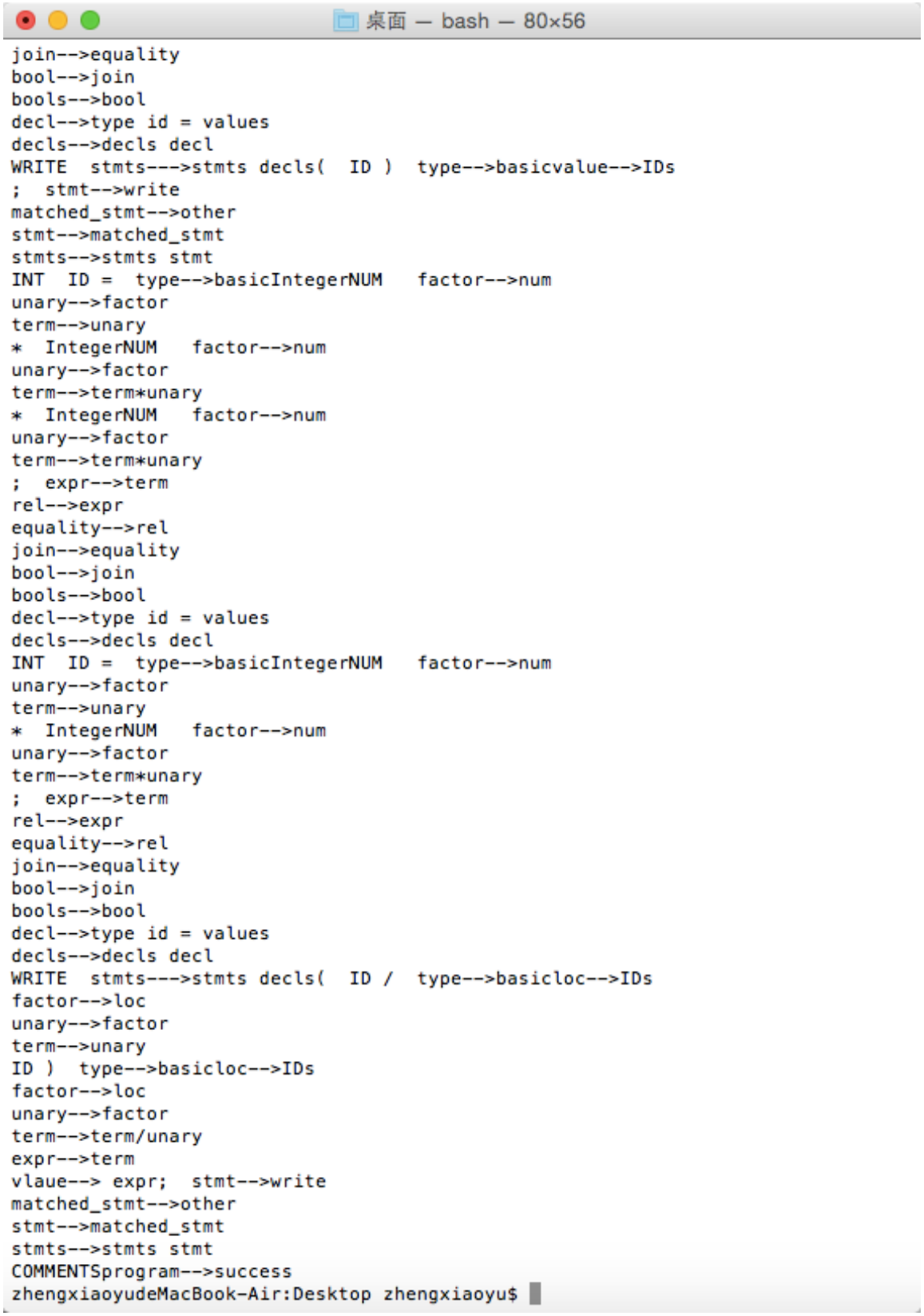
There are huge number of those pictures, in order to save pages, I just show the end of the output for each file.



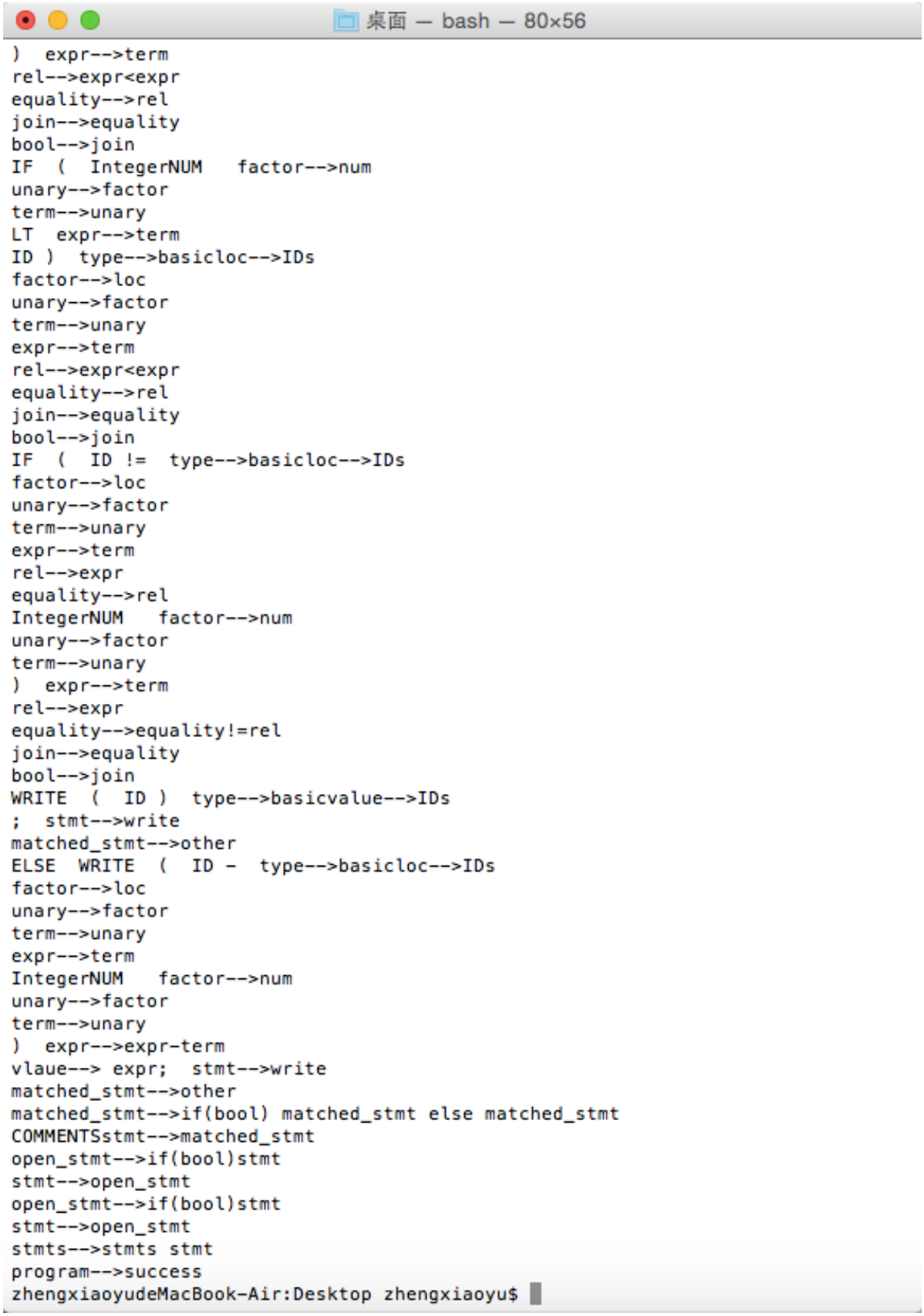
the result is success for test1.txt

successful for test2.txt

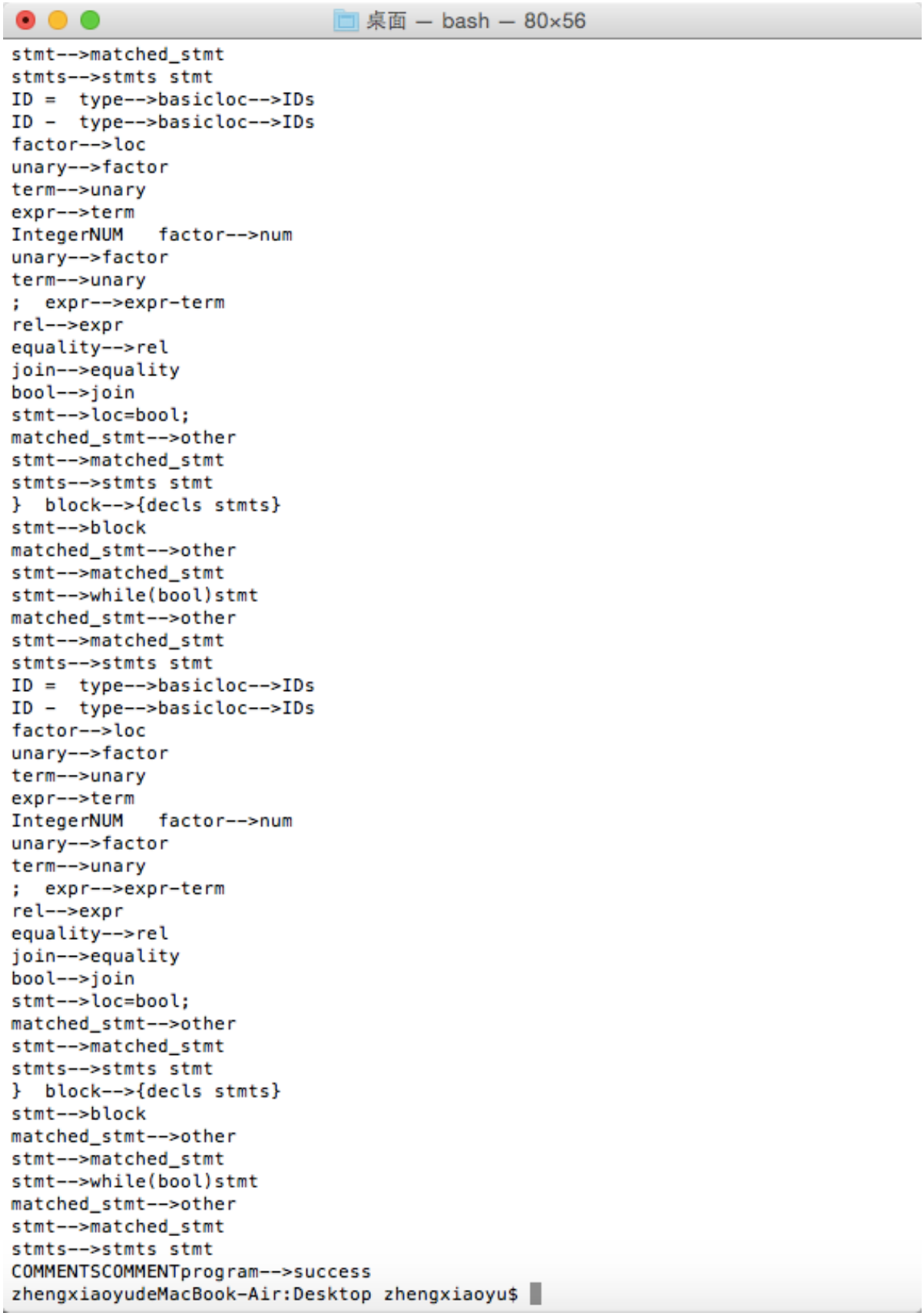
successful for test3.txt



successful for test4.txt



successful for test5.txt



test6.txt successful

test7.txt sccessful

test8.txt successful

test9.txt also successful

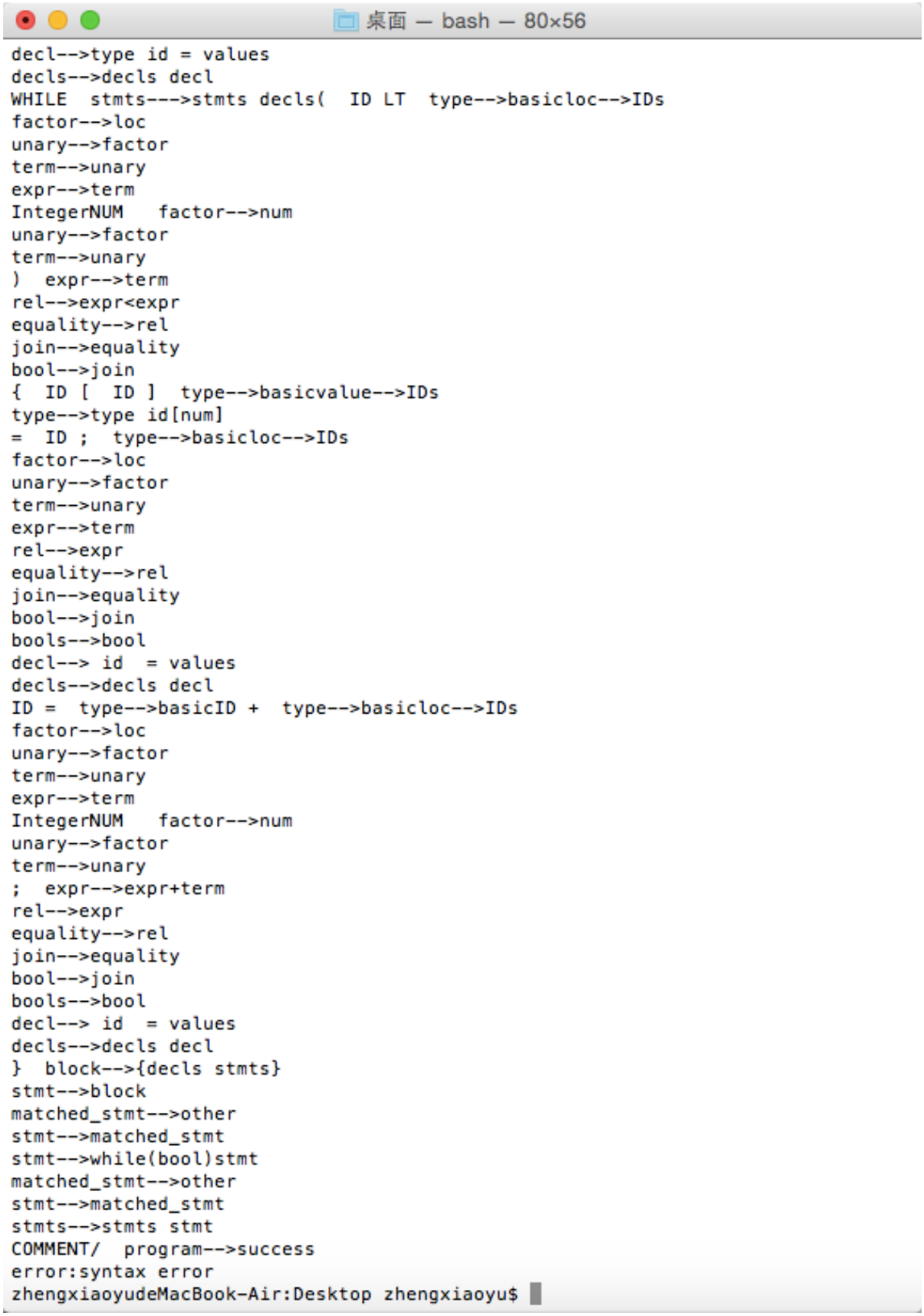
Then we test the error files

```
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$ ./parser error1_ID.txt
COMMENTSCOMMENTINT  ID ;  type-->basicdecl-->type id
decls-->decls decl
error:syntax error
real(double)  IntegerNUM   zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$
```

show error for error1.txt

```
桌面 — bash — 80×56
bool-->join
{  ID [  ID ]  type-->basicvalue-->IDs
type-->type id[num]
=  ID ;  type-->basicloc-->IDs
factor-->loc
unary-->factor
term-->unary
expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool
decl--> id  = values
decls-->decls decl
ID =  type-->basicID +  type-->basicloc-->IDs
factor-->loc
unary-->factor
term-->unary
expr-->term
IntegerNUM   factor-->num
unary-->factor
term-->unary
;  expr-->expr+term
rel-->expr
equality-->rel
join-->equality
bool-->join
bools-->bool
decl--> id  = values
decls-->decls decl
}  block-->{decls stmts}
stmt-->block
matched_stmt-->other
stmt-->matched_stmt
stmt-->while(bool)stmt
matched_stmt-->other
stmt-->matched_stmt
stmts-->stmts stmt
COMMENTID [  IntegerNUM   ]  value-->integer
type-->type id[num]
=  loc-->IDs
IntegerNUM   factor-->num
unary-->factor
term-->unary
;  expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
stmt-->loc=bool;
matched_stmt-->other
stmt-->matched_stmt
stmts-->stmts stmt
program-->success
zhengxiaoyudeMacBook-Air:Desktop zhengxiaoyu$
```

show no error in error2.txt, for it is not wrong in grammar in my parser



it shows error in error3.txt