

词法分析、语法分析程序实验：

18340146 计算机科学与技术 宋渝杰

实验题目

实验目的：扩充已有的样例语言 TINY，为扩展 TINY 语言 TINY+ 构造词法分析和语法分析程序，从而掌握词法分析和语法分析程序的构造方法。

实验内容：了解样例语言 TINY 及 TINY 编译器的实现，了解扩展 TINY 语言 TINY+，用 EBNF 描述 TINY+ 的语法，用 C 语言扩展 TINY 的词法分析和语法分析程序，构造 TINY+ 的语法分析器。

实验要求：将 TINY+ 源程序翻译成对应的 TOKEN 序列，并能检查一定的词法错误。将 TOKEN 序列转换成语法分析树，并能检查一定的语法错误。

实验过程

TINY 语言：

- 关键字：IF ELSE WRITE READ RETURN BEGIN END MAIN INT REAL
- 分割符：; , ()
- 一元运算符：+ - * /
- 多元运算符：:= == !=
- 标识符：标识符包含一个字母，后跟任意数量的字母或数字。以下是标识符的示例：x, x2, xx2, x2x, End, END2。请注意，End 是标识符，而 END 是关键字。以下不是标识符：
 - IF, WRITE, READ, ... (关键字不算作标识符)
 - 2x (标识符不能以数字开头)
 - 注释中的字符串不是标识符
- 数字：是一个数字序列，或一个数字序列，后跟一个小数点，再跟一个数字序列

```
Number -> Digits | Digits '.' Digits
Digits -> Digit | Digit Digits
Digit  -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

- 注释：/** 和 */之间的字符串，注释可以超过一行。

关于它的完全定义可以参考官网 <https://jlu.myweb.cs.uwindsor.ca/214/language.htm>

EBNF 文法

高级程序结构：

```
Program -> MethodDecl MethodDecl*
MethodDecl -> Type [MAIN] Id '(' FormalParams ')' Block
FormalParams -> [FormalParam ( ',' FormalParam )* ]
FormalParam -> Type Id

Type -> INT | REAL
```

声明:

```
Block -> BEGIN Statement* END

Statement -> Block
           | LocalVarDecl
           | AssignStmt
           | ReturnStmt
           | IfStmt
           | WriteStmt
           | ReadStmt

LocalVarDecl -> INT Id ';'
              | REAL Id ';'

AssignStmt -> Id := Expression ';'
ReturnStmt -> RETURN Expression ';'
IfStmt -> IF '(' BoolExpression ')' Statement
        | IF '(' BoolExpression ')' Statement ELSE Statement
WriteStmt -> WRITE '(' Expression ',' QString ')' ';'
ReadStmt -> READ '(' Id ',' QString ')' ';'
QString is any sequence of characters except double quote itself, enclosed in double quotes.
```

表达:

```
Expression -> MultiplicativeExpr (( '+' | '-' ) MultiplicativeExpr)*
MultiplicativeExpr -> PrimaryExpr (( '*' | '/' ) PrimaryExpr)*
PrimaryExpr -> Num // Integer or Real numbers
              | Id
              | '(' Expression ')'
              | Id '(' ActualParams ')'
BoolExpression -> Expression '==' Expression
                | Expression '!=' Expression
ActualParams -> [Expression ( ',' Expression)*]
```

TINY+

在上述 TINY 语法上，我添加了一些关键字和运算符，构建更加全面的 TINY+ 语法

- 关键字: **WHILE FOR**
- 一元运算符: % (mod), ^ (xor)
- 多元运算符: > <

TINY+ 的 EBNF 文法

以下仅列出有修改的部分:

声明:

```

Statement -> Block
          | LocalVarDecl
          | AssignStmt
          | ReturnStmt
          | IfStmt
          | WriteStmt
          | ReadStmt
          | WhileStmt
          | ForStmt

WhileStmt -> WHILE '(' BoolExpression ')' Statement
ForStmt -> For '(' AssignStmt ';' BoolExpression ';' AssignStmt ')' Statement

```

表达:

```

Expression -> AdditiveExpression ( '^' AdditiveExpression ) * // 位异或符号优先级最低
AdditiveExpression -> MultiplicativeExpression (( '+' | '-' )
MultiplicativeExpression)*
MultiplicativeExpression -> PrimaryExpr (( '*' | '/' ) PrimaryExpr)*
BoolExpression -> Expression '==' Expression
                | Expression '!=' Expression
                | Expression '>' Expression
                | Expression '<' Expression

```

词法分析:

词法分析的主要内容为：输入一个字符串，识别出它属于什么类型。识别的具体规则如下：

- 关键字：一类特殊的字符串，通过遍历所有的关键字，判断字符串是否相等即可
- 分隔符：单独识别出字符为 `;`, `(`, `)` 即可
- 运算符：单独识别出 `+`, `-`, `*`, `/`, `:=`, `==`, `!=`, `>`, `<`, `%`, `^` 即可
- ID（变量名/函数名）：一类只包含数字和字母的字符串，且不以数字开头、不为关键字
- 数字：一个数字序列，或一个数字序列 `!` 一个数字序列
- 注释： `/**` 和 `**/` 之间的任意字符串

具体程序识别的过程可以通过 DFA 来实现，设定初始状态，之后分析下一个字符：

- 初始状态：DFA 识别的初始状态
 - 数字：转换为“整数状态”
 - 字符：转换为“ID 状态”
 - 引号：转换为“字符串状态”（文件名字符串）
 - 冒号：转换为“赋值状态”
 - 等号：转换为“相等状态”
 - 感叹号：转换为“不等状态”
 - 左斜杠（且再下两位为乘号）：转换为“注释状态”
 - EOF：结束，返回 EOF
 - 运算符/分隔符：结束，返回对应符号
 - 空格、换行符、tab 符：无视
 - 其它：Error
- 赋值状态：识别赋值符号 `:=`
 - 等号：识别成功，结束，返回赋值符号 `:=`

- 其它: Error
- 相等状态: 识别相等符号 ==
 - 等号: 识别成功, 结束, 返回相等符号 ==
 - 其它: Error
- 不等状态: 识别不等符号 !=
 - 等号: 识别成功, 结束, 返回不等符号 !=
 - 其它: Error
- 注释状态: 识别注释结束符号 **/
 - 星号 (且再下两位为星号和右斜杠): 识别成功, 返回初始状态
 - 其它: 继续注释状态
- ID 状态: 识别 ID 结束
 - 字母 or 数字: 继续 ID 状态
 - 其它: ID 识别成功, 结束, 判断是否为关键字, 是则返回关键字, 否则返回变量名/函数名
- 整数状态: 识别数字结束
 - 小数点: 转换为“小数状态”
 - 数字: 继续整数状态
 - 其它 (非字母): 整数识别成功, 结束, 返回整数值
 - 字母: Error
- 小数状态: 识别数字结束
 - 数字: 继续小数状态
 - 其它 (非字母): 小数识别成功, 结束, 返回小数值
 - 字母: Error
- 字符串状态: 识别文件名结束
 - 引号: 文件名识别成功, 结束, 返回文件名字符串
 - 其它: 继续字符串状态

代码实现可以参考 src/lex.h 的 getToken 函数:

```
TokenType getToken();
```

输出 TOKEN 序列: getToken 函数返回识别出的词法之后, 直接输出即可

```
void printToken(TokenType token, char* str);
```

检查词法错误: 由上文 DFA 可知, getToken 函数可以识别出词法错误, 之后调用此函数输出错误信息

```
static void lexError();
```

语法分析:

语法分析的主要内容: 词法分析识别出所有词语类型之后, 判断词语之间的关联是否满足要求, 具体规则为:

- 程序由一个或多个函数构成
- 函数由返回类型、[MAIN 关键字]、函数名、左括号、[形参列表]、右括号、代码块组成
- 形参列表由逗号分隔的零个、一个或多个形参组成
- 形参由参数类型、参数名组成
- 代码块由 BEGIN 关键字、一个或多个声明、END 关键字组成

- 声明为代码块、变量声明、赋值语句、返回语句、IF 语句、WRITE 语句、READ 语句、WHILE 语句、FOR 语句其中之一
- 变量声明由变量类型、变量名、分号，或者变量类型、赋值语句组成
- 赋值语句由变量名、赋值符号、表达式、分号组成
- 返回语句由 RETURN 关键字、表达式、分号组成
- IF 语句由 IF 关键字、左括号、判断式、右括号、声明、[ELSE 关键字、声明] 组成
- WRITE 语句由 WRITE 关键字、左括号、表达式、逗号、文件名字符串、右括号、分号组成
- READ 语句由 READ 关键字、左括号、变量名、逗号、文件名字符串、右括号、分号组成
- 文件名字符串由引号、任意字符串、引号组成
- WHILE 语句由 WHILE 关键字、左括号、判断式、右括号、声明组成
- FOR 语句由 FOR 关键字、左括号、赋值语句、分号、判断式、分号、赋值语句、右括号、声明组成
- 表达式由加性表达式、零个或多个“位异或运算符和加性表达式”组成
- 加性表达式由乘性表达式、零个或多个“加/减运算符和乘性表达式”组成
- 乘性表达式由单一表达式、零个或多个“乘/除运算符和单一表达式”组成
- 单一表达式为数字、变量名、括号括起的表达式、函数调用之一
- 括号括起的表达式由左括号、表达式、右括号组成
- 函数调用由函数名、左括号、实参列表、右括号组成
- 判断式由表达式、判断符号、表达式组成
- 实参列表由逗号分隔的零个、一个或多个表达式组成

因此，分析过程可以通过如下描述：初始化为程序开始，按顺序识别一个或多个函数；对于识别函数，按顺序识别返回类型、[MAIN 关键字]、函数名、左括号、[形参列表]、右括号、代码块；而对于代码块，按顺序识别BEGIN 关键字、一个或多个声明、END 关键字...

代码实现可以参考 src/syntax.h 的 syntax 函数：

```
struct TreeNode* syntax();
```

语法分析树的节点：由上文得知：“一个类型”由“另一个或一些类型”组成，因此可以定义“一个类型”节点，它的子节点为“另一个或一些类型”节点。而一些子类型可以是不定长的（比如说程序由不定长个函数构成），此时定义一个节点链表，链表的头节点作为子节点，而其它节点则依次链接在头节点的后面。代码具体定义如下：

```
struct TreeNode {
    struct TreeNode* child[10];           // 子节点
    struct TreeNode* next;                // 指向链表中下一个节点
    NodeKind nodekind;                    // 节点类型
    struct Kind kind;                     // 节点的一些性质
    struct Attr attr;
};
```

节点的类型：主要定义了五种类型的节点：函数、类型、参数、声明、表达式。

- 函数包括 main 函数、其它函数
- 类型包括函数返回类型、INT 类型、REAL 类型
- 参数包括形参、实参
- 声明包括 IF 语句、RETURN 语句、赋值语句、READ 语句、WRITE 语句等
- 表达式包括符号、数字、变量、函数调用等

```
typedef enum { MethodK, TypeK, ParamK, StatementK, ExpressionK } NodeKind;
typedef enum { MainK, NormalK } MethodKind;
typedef enum { ReturnTypeK, IntTypeK, RealTypeK } TypeKind;
typedef enum { FormalK, ActualK } ParamKind;
typedef enum { IfK, ReturnK, AssignK, ReadK, WriteK, IntDeclareK, RealDeclareK,
whileK, Fork } StatementKind;
typedef enum { OpK, ConstK, IdK, MethodCallK } ExpressionKind;
```

分析树的构建：以“程序由一个或多个函数构成”为例，由于函数子节点不定长，因此构建一个节点链表，每分析到一个函数，就新建一个函数节点，并接在链表后面，最后返回链表头部，成为父节点的子节点

```
static struct TreeNode* MethodSequence() { // 函数节点序列
    struct TreeNode* t = Method();
    struct TreeNode* p = t;
    while (token != ENDFILE) {
        struct TreeNode* q = Method();
        p->next = q;
        p = q;
    }
    return t;
}
```

而对于“赋值语句由变量名、赋值符号、表达式、分号组成”为例，赋值语句节点固定，这里使用父节点（声明节点）记录赋值语句的变量名部分，而子节点则记录表达式部分

```
static struct TreeNode* AssignStatement() { // 赋值节点
    struct TreeNode* t = newStatementNode(AssignK);
    if (token == ID) t->attr.name = copyString(tokenString);
    match(ID); // ID := 表达式 ;
    match(ASSIGN);
    t->child[0] = Expression();
    return t;
}
```

分析树的输出：先序遍历即可，值得注意的是在深搜的时候，需要记录此时的深度，以便在输出时输出适当的空格以显示出树的层次结构

```
void printTree(struct TreeNode* root);
```

检查语法错误：根据上文对规则的定义，如果出现了匹配错误（如函数开头匹配不到函数类型），则返回匹配错误，之后调用此函数输出错误信息

```
static void syntaxError(char* errorMessage);
```

实验结果

词法分析：对两个 tiny 文档进行词法分析，两个文档内容分别如下：

test1: (txt/test1.tiny) , TINY 的基本内容

```

/** this is a comment line in the sample program */
INT f2(INT x, INT y )
BEGIN
    INT z;
    z := x*x - y*y;
    RETURN z;
END
INT MAIN f1()
BEGIN
    INT x;
    READ(x, "A41.input");
    INT y;
    READ(y, "A42.input");
    INT z;
    z := f2(x,y) + f2(y,x);
    WRITE (z, "A4.output");
END

```

test2: (txt/test2.tiny) , TINY+ 的基本所有内容

```

/** this is another sample program */
INT f1(INT x)
BEGIN
    INT i := 0;
    REAL ans;
    FOR (i := 0; i < x; i := i + 1)
        ans := ans + i;
    RETURN ans;
END
INT f2(INT x, INT y )
BEGIN
    INT z;
    z := x*x - y/y;
    IF (x != y) z := z + x; ELSE z := z - x;
    RETURN z;
END
INT MAIN f()
BEGIN
    INT x;
    READ(x, "A41.input");
    INT y;
    READ(y, "A42.input");
    INT z;
    z := f1(x) + f2(y,x);
    WHILE (x > y)
    BEGIN
        z := z ^ x % y;
        x := x - y;
    END
    WRITE (z, "A4.output");
END

```

程序的结果如下:

test1.tiny:

```
----- WELCOME TO COMPILER -----
1. Lexical analysis test1
2. Lexical analysis test2
3. Syntax analysis test1
4. Syntax analysis test2
0. Exit
-----
[INPUT] Please enter 0-4 ^_^
1
( KEYWORD, INT )
( ID, f2 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )
```

输出的所有内容：（注释并不输出）

```
( KEYWORD, INT )
( ID, f2 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )
( SEP, , )
( KEYWORD, INT )
( ID, y )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )
( ID, z )
( SEP, ; )
( ID, z )
( OP, := )
( ID, x )
( OP, * )
( ID, x )
( OP, - )
( ID, y )
( OP, * )
( ID, y )
( SEP, ; )
( KEYWORD, RETURN )
( ID, z )
( SEP, ; )
( KEYWORD, END )
( KEYWORD, INT )
( KEYWORD, MAIN )
( ID, f1 )
( SEP, ( )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )
( ID, x )
( SEP, ; )
( KEYWORD, READ )
( SEP, ( )
( ID, x )
( SEP, , )
```



```

( STR, "A41.input" )
( SEP, ) )
( SEP, ; )
( KEYWORD, INT )
( ID, y )
( SEP, ; )
( KEYWORD, READ )
( SEP, ( )
( ID, y )
( SEP, , )
( STR, "A42.input" )
( SEP, ) )
( SEP, ; )
( KEYWORD, INT )
( ID, z )
( SEP, ; )
( ID, z )
( OP, := )
( ID, f2 )
( SEP, ( )
( ID, x )
( SEP, , )
( ID, y )
( SEP, ) )
( OP, + )
( ID, f2 )
( SEP, ( )
( ID, y )
( SEP, , )
( ID, x )
( SEP, ) )
( SEP, ; )
( KEYWORD, WRITE )
( SEP, ( )
( ID, z )
( SEP, , )
( STR, "A4.output" )
( SEP, ) )
( SEP, ; )
( KEYWORD, END )

```

test2.tiny:

```

C:\Users\Song\Desktop\S & W大三下\编译原理\编译原理3\mine\src\main.exe
----- WELCOME TO COMPILER -----
1. Lexical analysis test1
2. Lexical analysis test2
3. Syntax analysis test1
4. Syntax analysis test2
0. Exit
-----
[INPUT] Please enter 0-4 ^_^
2
( KEYWORD, INT )
( ID, f1 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )

```

输出的所有内容：（注释并不输出）

```
( KEYWORD, INT )
( ID, f1 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )
( ID, i )
( OP, := )
( NUM, 0 )
( SEP, ; )
( KEYWORD, REAL )
( ID, ans )
( SEP, ; )
( KEYWORD, FOR )
( SEP, ( )
( ID, i )
( OP, := )
( NUM, 0 )
( SEP, ; )
( ID, i )
( OP, < )
( ID, x )
( SEP, ; )
( ID, i )
( OP, := )
( ID, i )
( OP, + )
( NUM, 1 )
( SEP, ) )
( ID, ans )
( OP, := )
( ID, ans )
( OP, + )
( ID, i )
( SEP, ; )
( KEYWORD, RETURN )
( ID, ans )
( SEP, ; )
( KEYWORD, END )
( KEYWORD, INT )
( ID, f2 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )
( SEP, , )
( KEYWORD, INT )
( ID, y )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )
( ID, z )
( SEP, ; )
( ID, z )
```

```
( OP, := )
( ID, x )
( OP, * )
( ID, x )
( OP, - )
( ID, y )
( OP, / )
( ID, y )
( SEP, ; )
( KEYWORD, IF )
( SEP, ( )
( ID, x )
( OP, != )
( ID, y )
( SEP, ) )
( ID, z )
( OP, := )
( ID, z )
( OP, + )
( ID, x )
( SEP, ; )
( KEYWORD, ELSE )
( ID, z )
( OP, := )
( ID, z )
( OP, - )
( ID, x )
( SEP, ; )
( KEYWORD, RETURN )
( ID, z )
( SEP, ; )
( KEYWORD, END )
( KEYWORD, INT )
( KEYWORD, MAIN )
( ID, f )
( SEP, ( )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )
( ID, x )
( SEP, ; )
( KEYWORD, READ )
( SEP, ( )
( ID, x )
( SEP, , )
( STR, "A41.input" )
( SEP, ) )
( SEP, ; )
( KEYWORD, INT )
( ID, y )
( SEP, ; )
( KEYWORD, READ )
( SEP, ( )
( ID, y )
( SEP, , )
( STR, "A42.input" )
( SEP, ) )
( SEP, ; )
```

```

( KEYWORD, INT )
( ID, z )
( SEP, ; )
( ID, z )
( OP, := )
( ID, f1 )
( SEP, ( )
( ID, x )
( SEP, ) )
( OP, + )
( ID, f2 )
( SEP, ( )
( ID, y )
( SEP, , )
( ID, x )
( SEP, ) )
( SEP, ; )
( KEYWORD, WHILE )
( SEP, ( )
( ID, x )
( OP, > )
( ID, y )
( SEP, ) )
( KEYWORD, BEGIN )
( ID, z )
( OP, := )
( ID, z )
( OP, ^ )
( ID, x )
( OP, % )
( ID, y )
( SEP, ; )
( ID, x )
( OP, := )
( ID, x )
( OP, - )
( ID, y )
( SEP, ; )
( KEYWORD, END )
( KEYWORD, WRITE )
( SEP, ( )
( ID, z )
( SEP, , )
( STR, "A4.output" )
( SEP, ) )
( SEP, ; )
( KEYWORD, END )

```

检测词法错误：将 test1.tiny 第四行从 INT z; 修改为 INT 1z; (变量名以数字开头，不合法)，测试结果如下：

C:\Users\Song\Desktop\S & W\大三下\编译原理\编译原理3\mine\src\main.exe

```
----- WELCOME TO COMPILER -----
1. Lexical analysis test1
2. Lexical analysis test2
3. Syntax analysis test1
4. Syntax analysis test2
0. Exit
-----
[INPUT] Please enter 0-4 ^_^
1
( KEYWORD, INT )
( ID, f2 )
( SEP, ( )
( KEYWORD, INT )
( ID, x )
( SEP, , )
( KEYWORD, INT )
( ID, y )
( SEP, ) )
( KEYWORD, BEGIN )
( KEYWORD, INT )

[LEXERROR] line 4, column 9 is wrong. Please check it >_<
请按任意键继续. . . ■
```

可以看出程序识别出词法错误，且成功定位在第四行第九列（即不合法变量名位置）

语法分析：对两个 tiny 文档进行语法分析，结果如下

test1.tiny:

C:\Users\Song\Desktop\S & W\大三下\编译原理\编译原理3\mine\src\main.exe

```
----- WELCOME TO COMPILER -----
1. Lexical analysis test1
2. Lexical analysis test2
3. Syntax analysis test1
4. Syntax analysis test2
0. Exit
-----
[INPUT] Please enter 0-4 ^_^
3
Method: f2
Return Type: INT
PARAM:
  INT: x
  INT: y
DECLARE: INT z
```

输出的所有内容：


```
Method: f2
Return Type: INT
PARAM:
  INT: x
  INT: y
DECLARE: INT z
ASSIGN: z
  OPERATOR: -
    OPERATOR: *
      ID: x
```

```

        ID: x
        OPERATOR: *
        ID: y
        ID: y
    RETURN
    ID: z
Main Method: f1
    Return Type: INT
    DECLARE: INT x
    READ "A41.input" to:
        ID: x
    DECLARE: INT y
    READ "A42.input" to:
        ID: y
    DECLARE: INT z
    ASSIGN: z
        OPERATOR: +
        FUNC: f2
        PARAM:
            ID: x
            ID: y
        FUNC: f2
        PARAM:
            ID: y
            ID: x
    WRITE "A4.output" from:
        ID: z

```

test2.tiny:

 C:\Users\Song\Desktop\S & W\大三下\编译原理\编译原理3\mine\src\main.exe

```

----- WELCOME TO COMPILER -----
        1. Lexical analysis test1
        2. Lexical analysis test2
        3. Syntax analysis test1
        4. Syntax analysis test2
        0. Exit
-----
[INPUT] Please enter 0-4 ^_^
4
    Method: f1
    Return Type: INT
    PARAM:
        INT: x
    DECLARE: INT i

```

输出的所有内容:

```

Method: f1
    Return Type: INT
    PARAM:
        INT: x
    DECLARE: INT i
    DECLARE: REAL ans
    FOR
        ASSIGN: i
        CONST: 0.000000
        OPERATOR: <

```

```

        ID: i
        ID: x
    ASSIGN: i
        OPERATOR: +
            ID: i
            CONST: 1.000000
    ASSIGN: ans
        OPERATOR: +
            ID: ans
            ID: i
    RETURN
        ID: ans
Method: f2
    Return Type: INT
    PARAM:
        INT: x
        INT: y
    DECLARE: INT z
    ASSIGN: z
        OPERATOR: -
            OPERATOR: *
                ID: x
                ID: x
            OPERATOR: /
                ID: y
                ID: y
    IF
        OPERATOR: !=
            ID: x
            ID: y
        ASSIGN: z
            OPERATOR: +
                ID: z
                ID: x
        ASSIGN: z
            OPERATOR: -
                ID: z
                ID: x
    RETURN
        ID: z
Main Method: f
    Return Type: INT
    DECLARE: INT x
    READ "A41.input" to:
        ID: x
    DECLARE: INT y
    READ "A42.input" to:
        ID: y
    DECLARE: INT z
    ASSIGN: z
        OPERATOR: +
            FUNC: f1
                PARAM:
                    ID: x
            FUNC: f2
                PARAM:
                    ID: y
                    ID: x


```

```

WHILE
  OPERATOR: >
    ID: x
    ID: y
  ASSIGN: z
    OPERATOR: ^
    ID: z
    OPERATOR: %
    ID: x
    ID: y
  ASSIGN: x
    OPERATOR: -
    ID: x
    ID: y
WRITE "A4.output" from:
  ID: z

```

检测语法错误：将 test1.tiny 第二行从 `INT f2(INT x, INT y)` 修改为 `f2(INT x, INT y)`（即去掉了函数返回类型），测试结果如下：

 C:\Users\Song\Desktop\S & W\大三下\编译原理\编译原理3\mine\src\main.exe

```

----- WELCOME TO COMPILER -----
1. Lexical analysis test1
2. Lexical analysis test2
3. Syntax analysis test1
4. Syntax analysis test2
0. Exit
-----
[INPUT] Please enter 0-4 ^_^
3

[SYNTAXERROR] line 2, column 3 is wrong: Invalid return type. Please check it >_<
请按任意键继续. . .

```