Tiny+编译器实现

相比于Tiny语言，我实现的Tiny+在原来的基础上新增加了全局变量GlobalVarDecl的声明，与局部变量LocalVarDecl的声明类似，如下：

```
LocalVarDecl -> Type Id ';' | Type AssignStmt
```

```
GlobalVarDecl -> Type Id ';' | Type AssignStmt
```

不同的是，LocalVarDecl必须在方法声明（MethodDecl）中的函数块（Block）中：

```
MethodDecl -> Type [MAIN] Id '(' FormalParams ')' Block

Block -> BEGIN Statement+ END

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

而GlobalVarDecl必须在Block外且在整个Program的开始，即：

```
Program -> GlobalVarDecl* MethodDecl MethodDecl*
GlobalVarDecl -> Type Id ';' | Type AssignStmt
```

综上，新的 Tiny+ Language 用EBNF语法表示如下：

The EBNF Grammar

High-level program structures

```
Program -> GlobalVarDecl* MethodDecl MethodDecl*
Type -> INT | REAL |STRING
MethodDecl -> Type [MAIN] Id '(' FormalParams ')' Block
FormalParams -> [FormalParam ( ',' FormalParam )* ]
FormalParam -> Type Id
```

Statements

```
Block -> BEGIN Statement+ END
GlobalVarDecl -> Type Id ';' | Type AssignStmt
Statement -> Block
           | LocalVarDecl
           | AssignStmt
           | ReturnStmt
           | IfStmt
           | WriteStmt
           | ReadStmt

LocalVarDecl -> Type Id ';' | Type AssignStmt

AssignStmt  -> Id := Expression ';'
           |  Id := QString ';'
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.
```

Expressions

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

Sample program

```
/** this is a comment line in the sample program **/

INT v1;
STRING v2 := "1234";

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;
    IF (x == y)
        z := f2(x, y) + f2(y, x);
    ELSE
        z := x + y;
    WRITE (z, "A4.output");

END
```

scanner

此部分为辅助程序文件浏览器，用于按字符读取tiny测试文件：包括读取下一行、读取下一字符以及回退至上一字符三个功能，为后面将Tiny语句转换为Token序列做准备

scanner结构：scanner通过row和column来定位某个字符

```
typedef struct scanner {
  FILE* file; //文件
  int row; //行数
  int column; //列数
} ScannerType;
```

nextLine函数：读取下一行，即将row加1, column重新置0

```
// 读取下一行。
void nextLine(ScannerType* scanner) {
  scanner->row++;
  scanner->column = 0;
}
```

nextChar函数：读取下一字符，column加1,通过fgetc函数将位置标识符向前移动，并将该位置的无符号char强制转换为int返回

```
// 读取下一字符。
int nextChar(ScannerType* scanner) {
  scanner->column++;
  return fgetc(scanner->file);
}
```

lastChar函数：回退至上一字符，column减1，通过ungetc函数将当前char放入文件中，变为下一个要读的字符

```
// 回退至上一字符。
void lastChar(ScannerType* scanner, char ch) {
  scanner->column--;
  ungetc(ch, scanner->file);
}
```

token

此部分定义了Tiny语句的token关键字类型，以及与对应Tiny语句符号的匹配函数，用于做词法分析 token类型:其中INT_LITERAL,REAL_LITERAL,STRING_LITERAL分别表示INT、REAL、STRING的字面量，EQ为判等，NE为不等，ASSIGN为赋值，IDENTIFIER为标识符

```
/* Tiny+ token type*/
typedef enum {
  /* special symbols */
  EQ = 256,NE,ASSIGN,IDENTIFIER,INT_LITERAL,REAL_LITERAL,STRING_LITERAL,

  /* keywords */
  INT,REAL,STRING,MAIN,BEGIN,END,IF,ELSE,READ,WRITE,RETURN
} TokenType;
```

matchKeyword函数：匹配关键字，传入从文件中读取的char*，判断其是否为某个关键字，返回其TokenType（注意一一对应）

```
/* Match keywords */
TokenType matchKeyword(const char* c) {
  for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); ++i) {
    if (strcmp(c, keywords[i]) == 0) {
      return INT + i;
    }
  }
  return IDENTIFIER;
}
```

lexical

此部分为Tiny语法的词法分析器，即将Tiny语句转换为Token序列，并能根据Token序列做一些简单的词法错误分析，需要注意的几点如下：

- 字符"/"的分析：因为该字符单独出现为运算符，而如果后面跟"*"，就可能为注释，所以这里需要做两种判断：若为运算符，直接返回"/"即可，如可能为注释，则需要判断注释的符号是否前后匹配

- 因为赋值运算符被定义为":="，而并非"="，所以当遇到字符"="，如果下一字符不为"="，即组合成为相等判别符，则为词法错误，所以这里不用做另外的判断
- 读取字符为"时，因为只有字符串字面量允许出现"，所以只需判断其后的字符集是否为字符串，同时检查末尾是否出现对应的"即可

Lex数据结构:

```
// 词法分析器。
typedef struct lex {
  ScannerType* scanner; // 文件浏览器
  char c;       // 当前输入字符
  char buf[256];        // 缓冲区：用于存储字面量
  char* p;      // 缓冲区指针
  char* literal;        // 字面量字符串
  int start;    // 记录起始位置
} LexType;
```

词法分析过程(源代码)：通过Scanner读取当前char：

- 若为空格和缩进，则直接跳过
- 若为换行符，则调用Scanner的nextLine函数跳到下一行
- 若为!、=、:，则调用Scanner的nextChar函数判断下一字符是否为=，若是，则返回相应的TokenType；若不是，则打印词法错误："Undefined symbol."
- 若为;、(、)、+、-、*、或者,，则直接返回字符
- 若为/，则根据前面注意的点中提到的判断其为运算符还是注释
- 若为"，读取后面的字符，判断是否符合字符串的定义
- 若为数字或者字母，则不断读取后面的字符，组成相应的字面量，储存到缓冲区中

parser

此部分为Tiny的语法分析器，是在词法分析的基础上将Token序列组合成各类语法短语，包括Program、Statements、Expressions等（根据EBNF语法），并能做一些简单的语法分析，最终通过ast(抽象语法树)的形式打印到文件中，用缩进数目来抽象表示树的父子关系

parser数据结构:

```
// 语法分析器
typedef struct parser {
  FILE* output; // 输出文件
  LexType* lex; // 词法分析器
  TokenType token;      // 当前分析字符
  int indentCounts;     // 缩进的数目
}ParserType;
```

语法分析过程(源代码)：通过Lex读取当前Token,再通过调用词法分析器中的nextToken函数将Tiny+程序转换为Token序列，再根据EBNF语法，用递归下降分析整个程序，需要分析的语法结构大致如下：

```
/* 根据 EBNF Grammar 对程序进行语法分析，构造语法树 */
void Program(ParserType* parser); //Tiny程序
void MethodDecl(ParserType* parser); //方法声明
void GlobalVarDecl(ParserType* parser); //全局变量声明
void Type(ParserType* parser); //类型
void Id(ParserType* parser); //名称
void FormalParams(ParserType* parser); //形参
void FormalParam(ParserType* parser); //具体形参
void Block(ParserType* parser); //函数块
void Statement(ParserType* parser); //语句
void LocalVarDecl(ParserType* parser); //局部变量
void AssignStmt(ParserType* parser); //赋值语句
void ReturnStmt(ParserType* parser); //Return语句
void IfStmt(ParserType* parser); //IF语句
void WriteStmt(ParserType* parser); //Write语句
void ReadStmt(ParserType* parser); //Read语句
void Expression(ParserType* parser); //表达式
void MultiplicativeExpr(ParserType* parser); //乘法表达式
void PrimaryExpr(ParserType* parser); //基本表达式
void ActualParams(ParserType* parser); //实际参数
void BoolExpression(ParserType* parser); //bool表达式
```

错误处理函数parser_error:

```
// 打印语法分析错误
void parse_error(ParserType* parser, int n, ...) {
  fprintf(stderr, "Line %d, Pos %d: Need ", SCANNER->row, LEX->start);
  va_list ex;
  va_start(ex, n);
  for (int i = 0; i < n; ++i) {
    if (i) {
      fprintf(stderr, " or ");
    }
    fprintf(stderr, "\"%s\"", getToken(va_arg(ex, int)));
  }
  va_end(ex);
  fprintf(stderr, ", but got \"%s\".\n", getToken(TOKEN));
  exit(1);
}
```

例子程序的抽象语法树结果如下：

```
Program
        ->GlobalVarDecl
                ->Type
                        ->INT
                ->Id
                        ->globalVar1
        ->GlobalVarDecl
```

```
                ->Type
                        ->STRING
                ->AssignStmt
                        ->Id
                                ->globalVar2
                        ->:=
                        ->STRING_LITERAL
                                ->"Tiny+ Compiler!"
        ->MethodDecl
                ->Type
                        ->INT
                ->Id
                        ->f2
                ->formal_params
                        ->FormalParams
                                ->Type
                                        ->INT
                                ->Id
                                        ->x
                        ->FormalParams
                                ->Type
                                        ->INT
                                ->Id
                                        ->y
                ->Block
                        ->Statement
                                ->LocalVarDecl
                                        ->Type
                                                ->INT
                                        ->AssignStmt
                                                ->Id
                                                        ->a
                                                ->:=
                                                ->Expression
                                                        ->MultiplicativeExpr
                                                                ->PrimaryExpr
                                                                        ->Id
                                                                                -
>x
                                                        ->+
                                                        ->MultiplicativeExpr
                                                                ->PrimaryExpr
                                                                        ->Id
                                                                                -
>y
                        ->Statement
                                ->LocalVarDecl
                                        ->Type
                                                ->INT
                                        ->Id
                                                ->z
                        ->Statement
                                ->AssignStmt
                                        ->Id
```

```
                                                ->z
                                        ->:=
                                        ->Expression
                                                ->MultiplicativeExpr
                                                        ->PrimaryExpr
                                                                ->Id
                                                                        ->x
                                                        ->*
                                                        ->PrimaryExpr
                                                                ->Id
                                                                        ->x
                                                ->-
                                                ->MultiplicativeExpr
                                                        ->PrimaryExpr
                                                                ->Expression
                                                                        -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr

->Id

->a
                                                                        ->-
                                                                        -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr

->Id

->y
                                                        ->*
                                                        ->PrimaryExpr
                                                                ->Id
                                                                        ->y
                        ->Statement
                                ->ReturnStmt
                                        ->Expression
                                                ->MultiplicativeExpr
                                                        ->PrimaryExpr
                                                                ->Id
                                                                        ->z
        ->MethodDecl
                ->Type
                        ->INT
                ->MAIN
                ->Id
                        ->f1
                ->formal_params
                ->Block
                        ->Statement
                                ->LocalVarDecl
                                        ->Type
```

```
                                        ->INT
                                ->Id
                                        ->x
                        ->Statement
                                ->ReadStmt
                                        ->Id
                                                ->x
                                        ->STRING_LITERAL
                                                ->"A41.input"
                        ->Statement
                                ->LocalVarDecl
                                        ->Type
                                                ->INT
                                        ->Id
                                                ->y
                        ->Statement
                                ->ReadStmt
                                        ->Id
                                                ->y
                                        ->STRING_LITERAL
                                                ->"A42.input"
                        ->Statement
                                ->LocalVarDecl
                                        ->Type
                                                ->INT
                                        ->Id
                                                ->z
                        ->Statement
                                ->IfStmt
                                        ->IF
                                                ->BoolExpression
                                                        ->Expression
                                                                -
>MultiplicativeExpr
                                                                        -
>PrimaryExpr
                                                                                -
>Id

->x
                                                        ->==
                                                        ->Expression
                                                                -
>MultiplicativeExpr
                                                                        -
>PrimaryExpr
                                                                                -
>Id

->y
                                                ->Statement
                                                        ->AssignStmt
                                                                ->Id
                                                                        ->z
```

```
                                                                    ->:=
                                                                    ->Expression
                                                                            -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr
->Id
->f2
->ActualParams
->Id
->x
->Expression
->MultiplicativeExpr
->PrimaryExpr
->Expression
->MultiplicativeExpr
->PrimaryExpr
->Id
->y
                                                                            ->+
                                                                            -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr
->Id
->f2
->ActualParams
->Id
->y
->Expression
->MultiplicativeExpr
->PrimaryExpr
```

```
->Expression

->MultiplicativeExpr

->PrimaryExpr

->Id

->x
                                        ->ELSE
                                                ->Statement
                                                        ->AssignStmt
                                                                ->Id
                                                                        ->z
                                                                ->:=
                                                                ->Expression
                                                                        -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr

->Id

->x
                                                                        ->+
                                                                        -
>MultiplicativeExpr
                                                                                -
>PrimaryExpr

->Id

->y
                        ->Statement
                                ->WriteStmt
                                        ->Expression
                                                ->MultiplicativeExpr
                                                        ->PrimaryExpr
                                                                ->Id
                                                                        ->z
                                        ->STRING_LITERAL
                                                ->"A4.output"
```

词法错误处理

由于这次是简单实现，所以错误只是逐条显示

- 出现未定义字符(组)类型，如"="、"<"

```
21      IF (x < y)
22          z := f2(x, y) + f2(y, x);
23      ELSE
24          z := x + y;
25      WRITE (z, "A4.output");
26  END
27
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 21, Col 11: Undefined symbol.
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

```
23      ELSE
24          z = x + y;
25      WRITE (z, "A4.output");
26  END
27
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 21, Col 11: Undefined symbol.
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 24, Col 12: Undefined symbol.
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

- 字符串格式错误（"未匹配）

```
3   INT globalVar1;
4   STRING globalVar2 := "Tiny+ Compiler!;
5
6   INT f2(INT x, INT y)
7   BEGIN
8       INT a := x + y;
9       INT z;
10      z := x * x - (a - y) * y;
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 4, Col 39: String symbols don't match.
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

语法错误处理

- 正式程序（除注释）未由Type(INT、STRING、REAL)开始，即非GlobalVarDecl和MethodDecl

```
1   /** this is a comment line in the sample program **/
2
3   IF globalVar1;
4   STRING globalVar2 := "Tiny+ Compiler!";
5
6   INT f2(INT x, INT y)
7   BEGIN
8       INT a := x + y;
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
lucky1@ubuntu:~/Desktop/Tiny+_$ MAKE
MAKE: command not found
lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 3, Pos 1: Need "INT" or "REAL", but got "IF".
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

- 方法块Block缺少BEGIN或者END

```
14  INT MAIN f1()
15      INT x;
16      READ(x, "A41.input");
17      INT y;
18      READ(y, "A42.input");
19      INT z;
20      IF (x == y)
21          z := f2(x, y) + f2(y, x);
22      ELSE
23          z := x + y;
24      WRITE (z, "A4.output");
25  END
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
lucky1@ubuntu:~/Desktop/Tiny+_$ make
lucky1@ubuntu:~/Desktop/Tiny+_$ make clean
lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 15, Pos 5: Need "BEGIN", but got "INT".
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

- ELSE前面没有IF语句

```
19      READ(y, "A42.input");
20      INT z;
21
22      ELSE
23          z := x + y;
24      WRITE (z, "A4.output");
25  END
26
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 22, Pos 5: Need "INT" or "REAL" or "IDENTIFIER" or "BEGIN" or "RETURN" or "IF" or "READ" or "WRITE", but got "ELSE".
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

- 缺少"("或者")"

```
25      WRITE z, "A4.output");
26  END
27
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 25, Pos 11: Need "(", but got "IDENTIFIER".
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

```
25      WRITE (z, "A4.output";
26  END
27
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

lucky1@ubuntu:~/Desktop/Tiny+_$ make
Line 25, Pos 26: Need ")", but got ";".
make: *** [parser] Error 1
lucky1@ubuntu:~/Desktop/Tiny+_$
```

源码地址