编译原理课程设计开发报告

第 2 组

1. 小组成员信息

姓名	学号	班级	邮箱	是否组长
许耀中	12330358	数媒 2 班	4552844@qq.com	1
杨柳	12330366	数媒 2 班	1151730511@qq.com	0
张剑涛	12330400	数媒 2 班	1131484815@qq.com	0
张培杰	12330401	数媒 2 班	243497963@qq.com	0
易圣舟	12330379	数媒 2 班	360816499@qq.com	0

2. 开发环境

操作系统：linux
编译器：g++
程序语言：C/C++

3. 思路和设计

这次课程设计主要分为三部分，即词法分析、语法分析以及 SSQL 语句的执行。词法分析器是读入输入文件并输出一个个的词法单元供语法分析使用，语法分析器读入以由词法单元组成的流，当语法分析器成功解析一条 SSQL 语句就执行相对的动作，如 CREATE 语句就创建一个 Table 的对象并存储这个对象；INSERT 语句就找到表名执行插入操作，找不到表名就报错；DELETE 语句就找到表名执行删除操作，找不到表名就报错；SELECT 语句找到表名执行查询操作，找不到表名就报错。当语法分析器找到一个 EOF 的词法单元，语法分析就会结束。

类名	实现的功能
Lexer	词法分析类，scan 函数返回一个 token
Parser	语法分析类，对 SSQL 语句解析和执行
Token	Token 类，定义了一个 int 变量 tag 标识词法单元
Num	从 Token 类继承而来，定义了 int 变量 value 存放数字的值
Word	从 Token 类继承而来，定义了 string 变量 str 存放标识符
Table	内存数据库的表，用 vector<vector<int>>存放数据，并支持查询、删除、插入等操作
Statement	存放一条关系运算表达式的类
Tag	定义了词法单元对应的常量

4. 代码实现

(1) 词法单元类

Tag 类定义了一些词法单元的常量，如下图：

```
class Tag
{
    public:
        enum {
        CREATE = 256, TABLE = 257, INT = 258, PRIMARY = 259, DEFAULT = 260,
        KEY = 261, INSERT = 262, INTO = 263, VALUES = 264, DELETE = 265,
        FROM = 266, WHERE = 267, SELECT = 268, ID = 269, AND = 270,
        NE = 271, OR = 272, EQ = 273, GE = 274, LE = 275, NUM = 276
        };

};
```

Token 类定义了一个唯一标识词法单元的 int 变量 tag，如下图

```
class Token {
    public:
        int tag;
        Token(int t) {
            tag = t;
        }
        Token() {}
        virtual string toString(){
            char c = (char)tag;
            string str(1, c);
            return str;
        }
};
```

Num 类是数字的词法单元类，从 Token 类继承而来，里面定义了一个 int 变量 value 存放数字的值，如下图：

```
class Num : public Token
{
    public:
        int value;
        Num(int v) : Token(276){
            value = v;
        }
        virtual string toString(){
            char buf[20];
            sprintf(buf, "%d", value);
            string res(buf);
            return res;
        }
};
```

Word 类是标识符（如表名、列名）和关键字（如 CREATE）的词法单元类，从 Token 类继承而来，里面定义了一个 string 变量存放标识符，如下图

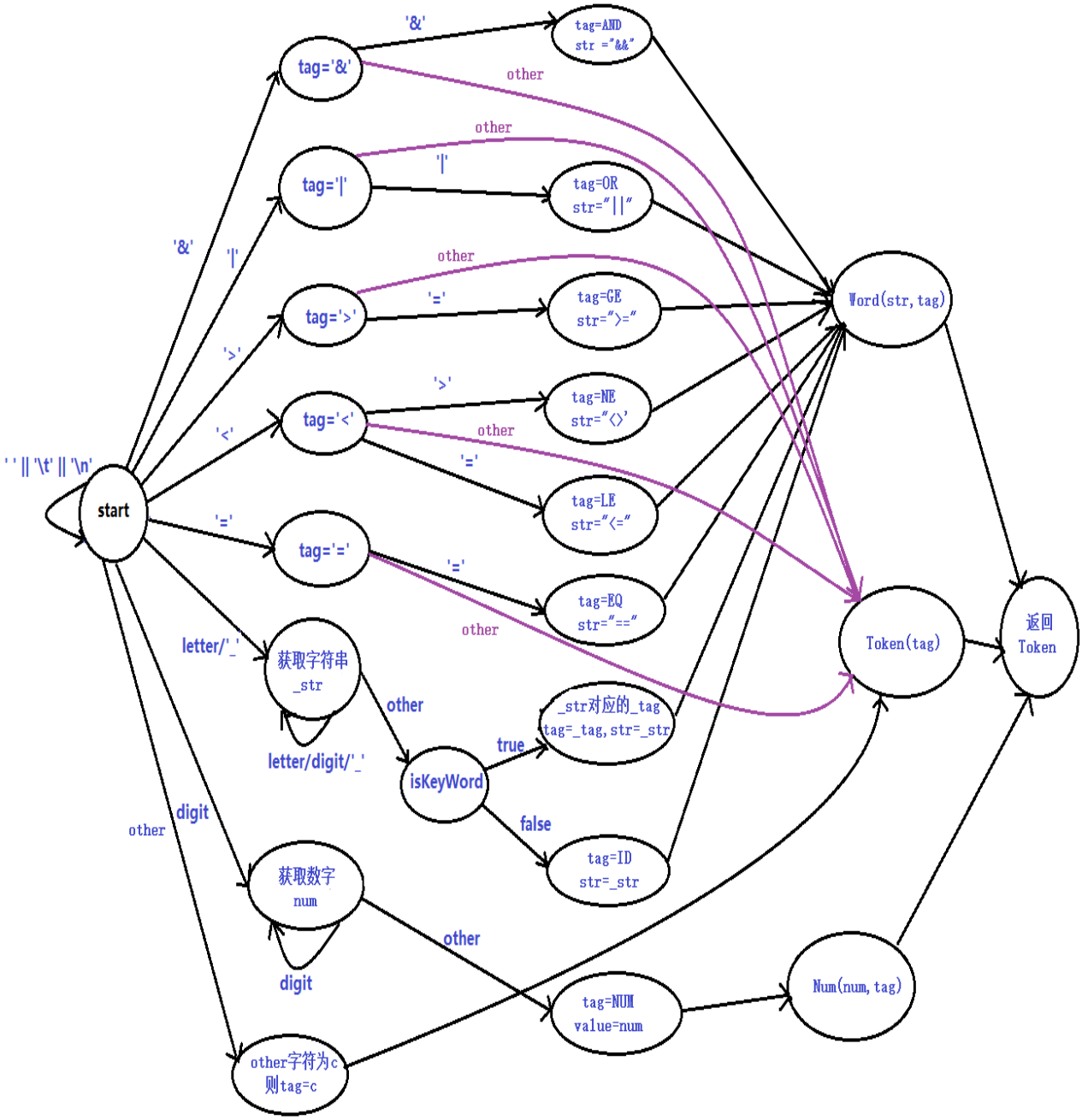
```
class Word : public Token
{
    public:
        string str;
        Word(string s, int tag) : Token(tag){
        str = s;
    }
    virtual string toString(){
        return str;
    }
};
```

(2) 词法分析类

词法分析类是由 Lexer 类实现，如下图，其中 scan 函数从输入流中读取一个词法单元供语法分析利用

```
class Lexer {
        public:
                Lexer();
                Token* scan();    //返回一个Token的指针
                void getChar();    //获取下一个字符
                bool getChar(char c);  //判断下一个字符是否为c
                int digit(char c);       //将字符c转化为数字
                bool isLetterOrUnderline(char c); //判断c是否为字母或下划线
                bool isDigit(char c);     //判断c是否为数字
                static int line;       //存放当前的行数
        private:
                char peek;    //存放当前的字符
                Tag tag;      //词法单元常量
                map<string, int> words;  //存放关键字以及对应的常量
};
```

词法分析主要是根据下图的 DFA 实现。



(3) 语法分析类

Parser 类实现语法分析的功能，里面定义了一个 map<string, Table>的变量 tables，里面存储了已经实例化的 Table 对象，每成功解析一条 SSQL 语句，语法分析器会从里面获取 Table 对象进行插入、插入以及删除操作。如果是创建操作，则实例化一个 Table 对象，然后将其放 tables 中。语法分析器的实现是基于递归的预测分析。

```
class Parser {
        private:
                Lexer lex;                      // lexical analyzer for this parser
                Token* look;                    // lookahead token
                Tag tag;
                map<string, Table> tables;      // 数据库实例
                string tabname;   //
                vector<string> colname;
                vector<string> pkey;
                map<string, int> defval;
                vector<int> val;
                vector<map<string, Statement*> > whereArg;
                map<string, Statement*> stmts;
                bool isall;

                Token* tmp_look;
                Word* tmp_word;
                Num* tmp_num;
                int res_cal;


        public:
                 Parser(Lexer l);

                 void Move();
                 bool Match(int t);
                 void Program();
                 bool CREATE_Rec();
                 bool Primary_Rec();
                 int INSERT_Col_Rec(int num);
                 int INSERT_Val_Rec(int num);
                 bool WHERE_Rec();
                 bool SELECT_Rec();

                 void Error(string s);
                 void Clear();

                 bool Calculator();
};
```

(4) Table 类实现

Table 类是实现的是数据库的一个表，支持创建、插入、查询和删除操作。里面定义了一个vector<vector<int>>的变量data存放具体的数据，空间复杂度为$o(n^2)$。由于 vector 是基于数字，可以根据下标获取数据，因此可以根据下标作为每一个列名的标识。对于插入操作，会检查是否存在主键冲突，因此要对主键的所有数据进行对比，时间复杂度为 o(n)。对于查询操作，所有的数据都有遍历一次，因此时间复杂度为 $o(n^2)$。对于删除操作，所有的数据都有遍历一次查看是否符合，因此时间复杂度也是 $o(n^2)$。

```
class Table {
    public:
        Table();
        Table(string tableName, vector<string> columnName, map<string, int>
defaultValue, vector<string> pkey);  //构造函数
        void delete_s(vector<map<string,Statement*> > whereArg); //删除
        void query(bool isAll, vector<string> column,
vector<map<string,Statement*> > whereArg);  //查询
        void insert(map<string,int> record);  //插入
    private:
        bool isRowDataRight(vector<int> row_data, vector<map<string,Statement*>
> whereArg);            //对一行数据进行匹配
        Statement dealingStmt(Statement stmt);  //对一个关系表达式进行处理
        string tableName;          //表名
        int columnCount;          //列的数量
        map<string, int> columnName;   //存储列名以及对应的下标
        map<string, int> defaultValue;  //有default值的列名和值
        vector<vector<int> > data;   //表的数据存储，根据下标获取数据
        vector<string> pkey;   //主键
};
```

5. 小组分工

姓名	负责的工作
许耀中	Table 类的实现
杨柳	语法分析器的实现以及测试
张剑涛	词法分析器的实现以及测试
张培杰	完成开发报告
易圣舟	完成测试报告