实验报告：基于Flex/Bison的简易DBMS系统设计与实现

一、实验目的与要求

1. 实验目的

- **加深编译原理基础知识的理解**：通过词法分析、语法分析、语法制导翻译等环节，理解编译器的基本原理。
- **加深数据库系统相关知识的理解**：通过实现数据库的基本操作，理解数据库系统、数据结构与操作系统的相关知识。

2. 实验要求

设计并实现一个DBMS原型系统，能够接受基本的SQL语句，对其进行词法分析、语法分析、语义分析，并解释执行SQL语句，实现对数据库文件的基本操作。
支持的SQL语句及功能如下：

SQL语句	功能
CREATE DATABASE	创建数据库
USE DATABASE	选择数据库
CREATE TABLE	创建表
SHOW TABLES	显示表名
INSERT	插入元组
SELECT	查询元组
UPDATE	更新元组
DELETE	删除元组
DROP TABLE	删除表
DROP DATABASE	删除数据库
EXIT	退出系统

> **注：** 支持数据类型有 `INT`、`CHAR(N)` 等。

二、实验环境配置

- **操作系统**：Windows 10/11
- **编译器**：GCC（MSYS2环境下的mingw-w64-x86_64-gcc）
- **工具链**：Flex、Bison
- **开发环境**：VSCode、PowerShell

环境安装步骤

1. **安装MSYS2**
   下载并安装 MSYS2。
2. **安装开发工具链**
   打开MSYS2 MINGW64终端，执行：

```
pacman -Syu
pacman -S mingw-w64-x86_64-gcc mingw-w64-x86_64-flex mingw-w64-x86_64-bison
```

```
3. **配置环境变量**
   将 `C:\msys64\mingw64\bin` 添加到系统环境变量 `PATH`。

4. **验证安装**
   在终端输入 `gcc --version`、`flex --version`、`bison --version`，输出版本号。

---

## 三、系统设计

### 1. 总体架构设计

本系统采用**模块化设计**，主要分为四大模块：

- **词法分析模块（Flex）**：负责将输入的SQL语句分解为Token流。
- **语法分析模块（Bison）**：根据SQL语法规则对Token流进行语法分析，构建语法树并驱动后续操作。
- **语义执行模块（C代码）**：在Bison的动作代码中实现具体的数据库操作逻辑。
- **数据存储与管理模块（C结构体链表）**：所有数据库、表、字段、记录等信息均存储于内存链表结构中。

#### 数据流转说明

1. 用户输入SQL语句
2. Flex将输入分词，生成Token
3. Bison根据Token流进行语法分析，匹配规则并调用相应C函数
4. C函数操作内存中的数据库/表/记录链表，实现SQL语义
5. 输出操作结果或错误信息

---

### 2. 主要数据结构设计

#### 2.1 数据库结构体

```c
struct mydb {
    char name[50];              // 数据库名
    struct table *tbroot;       // 指向该数据库下的表链表
    struct mydb *next;          // 下一个数据库
};
```

- 设计思路：采用链表管理多个数据库，每个数据库下挂载自己的表链表。
```

2.2 表结构体

```
struct table {
    char name[50];              // 表名
    struct field *ffield;       // 字段数组（每个字段包含所有记录的数据）
    int flen;                   // 字段数
    int ilen;                   // 记录数
    struct table *next;         // 下一个表
};
```

- 设计思路：每个表有自己的字段数组，字段数组中每个元素存储所有记录的该字段值。

2.3 字段结构体

```
struct field {
    char name[50];              // 字段名
    int type;                   // 0: int, 1: string
    struct key {
        int intkey;
        char skey[50];
    } key[100];                 // 最多100条记录
};
```

- 设计思路：每个字段有一个key数组，key[i]表示第i条记录在该字段的值。

2.4 其他结构体

- **item_def**：用于SELECT/UPDATE等语句的字段链表
- **hyper_items_def**：用于CREATE TABLE时的字段定义链表
- **value_def**：用于INSERT等语句的值链表
- **conditions_def**：用于WHERE条件的二叉树
- **table_def**：用于多表操作的表链表
- **upcon_def**：用于UPDATE语句的赋值链表

---

3. 主要功能实现细节

3.1 数据库管理

- **创建数据库**：`createDB()`
  检查数据库名是否已存在，若不存在则在dbroot链表尾部插入新数据库节点。
- **删除数据库**：`dropDB(char *dbname)`
  遍历dbroot链表，找到目标数据库，释放其所有表和自身内存。
- **切换数据库**：`useDB(char *dbname)`
  遍历dbroot链表，找到目标数据库，更新全局变量`database`。

3.2 表管理

- **创建表**：`createTable(char *tableval, struct hyper_items_def *Hitemroot)`
  在当前数据库下新建表节点，初始化字段数组，设置字段名和类型。
- **删除表**：`dropTable(char *tableval)`
  遍历表链表，找到目标表，释放其字段数组和自身内存。
- **显示表**：`showTable()`
  遍历当前数据库下的表链表，输出所有表名。

3.3 数据操作

- **插入记录**：`multiInsert(char *tableval, struct item_def *itemroot, struct value_def *valroot)`
  找到目标表，将值链表中的数据依次插入到字段数组的key数组中，支持指定字段插入和全字段插入。
- **查询记录**：`selectWhere(struct item_def *itemroot, struct table_def *tableroot, struct conditions_def *conroot)`
  支持单表和两表联合查询，遍历记录并根据WHERE条件筛选，输出指定字段。
- **更新记录**：`updates(struct table *tabletemp, struct upcon_def *uptemp, struct conditions_def *conroot)`
  遍历表的记录，若满足WHERE条件则将指定字段更新为新值。
- **删除记录**：`deletes(char *tableval, struct conditions_def *conroot)`
  遍历表的记录，若满足WHERE条件则删除该记录（通过后移覆盖实现）。

3.4 词法与语法分析

- **sql.l**：定义所有SQL关键字、标识符、数字、字符串的正则表达式，生成Token。
- **sql.y**：定义SQL语句的BNF文法，使用Bison的动作代码调用上述C函数，实现SQL语义。

---

4. 关键实现举例

4.1 CREATE TABLE 语句处理流程

1. 用户输入：`CREATE TABLE STUDENT(SNAME CHAR(20), SAGE INT, SSEX INT);`
2. Flex识别关键字、标识符、类型等Token
3. Bison匹配CREATE TABLE语法规则，构建hyper_items_def链表
4. 动作代码调用`createTable`，在当前数据库下新建表，初始化字段数组
5. 输出"Table STUDENT created successfully!"

4.2 SELECT 语句处理流程

1. 用户输入：`SELECT SNAME, SAGE FROM STUDENT WHERE SSEX = 1;`
2. Flex分词，Bison语法分析，构建item_def链表和conditions_def条件树
3. 动作代码调用`selectWhere`，遍历STUDENT表所有记录，筛选SSEX=1的记录，输出SNAME和SAGE字段

---

5. 内存管理与异常处理

- 所有链表节点和数组均动态分配内存，操作完成后及时释放，防止内存泄漏。
- 对所有输入参数和操作结果进行合法性检查，发现错误及时输出提示信息。

如需进一步细化某一部分（如具体函数实现、数据结构图示、流程图等），请随时告知！

四、实现细节

1. 词法分析（sql.l）

- 定义SQL关键字、标识符、数字、字符串等Token。
- 忽略空白字符和注释。

2. 语法分析（sql.y）

- 定义SQL语句的BNF文法规则。
- 每个语法规则对应C代码动作，实现具体操作。
- 通过`%union`和`%type`支持多种数据类型的语义值传递。

3. 主要C文件

- **sql.c**：实现所有数据库、表、记录的操作函数。
- **sql.h**：定义所有结构体和函数声明。
- **main函数**：在`sql.y`中，调用`yyparse()`启动交互式SQL解析。

4. 编译与运行

```
flex sql.l
bison -d sql.y
gcc -o mysql sql.tab.c lex.yy.c sql.c
./mysql
```

五、运行结果展示

```
SQL>CREATE DATABASE XJGL;
Database XJGL created successfully!

SQL>CREATE DATABASE JUST_FOR_TEST;
Database JUST_FOR_TEST created successfully!

SQL>CREATE DATABASE JUST_FOR_TEST;
error: The database already exists!

SQL>SHOW DATABASES;
XJGL  JUST_FOR_TEST

SQL>DROP DATABASE JUST_FOR_TEST;
Drop database successfully!

SQL>SHOW DATABASES;
XJGL

SQL>USE XJGL;
Using database XJGL

SQL>CREATE TABLE STUDENT(SNAME CHAR(20),SAGE INT,SSEX INT);
Table STUDENT created successfully!

SQL>CREATE TABLE COURSE(CNAME CHAR(20),CID INT);
Table COURSE created successfully!

SQL>SHOW TABLES;
```

```
STUDENT  COURSE
```

六、遇到的困难与解决办法

1. **编译报错（类型不匹配、未声明变量等）**
   - 解决：仔细检查头文件声明与实现是否一致，结构体成员是否正确，参数类型是否匹配。
2. **环境配置问题（找不到flex/bison/gcc）**
   - 解决：采用MSYS2环境，确保所有工具链都已正确安装并配置到PATH。
3. **SQL语法设计与冲突**
   - 解决：合理设计BNF文法，避免二义性，充分利用Bison的优先级和结合性声明。
4. **内存管理**
   - 解决：为每个链表、结构体分配和释放内存，防止内存泄漏。

七、实验反思

- **理论与实践结合**：通过本实验，深刻体会到编译原理知识在实际项目中的应用，尤其是词法、语法分析的自动化工具（Flex/Bison）的强大。
- **调试与排错能力提升**：遇到多次编译和运行时错误，通过查阅文档和调试逐步解决，提升了问题定位和解决能力。
- **代码规范与结构设计**：良好的结构体设计和代码规范极大提升了系统的可维护性和可扩展性。
- **团队协作与文档编写**：实验过程中注重文档记录和代码注释，为后续复现和扩展打下基础。

八、总结

实现了一个支持基本SQL语句的简易DBMS系统，涵盖了数据库、表、记录的基本操作。通过本次实验，不仅加深我了对编译原理和数据库系统的理解，也锻炼了实际动手能力和解决问题的能力。