好的，下面是编译原理词法分析器实验报告的正文部分示范内容，包含每部分内容和插图位置说明，你可以根据需要修改和补充：

# 编译原理词法分析器实验报告

## 一、实验目的

通过设计并实现一个词法分析器，掌握正规文法、正规式、有穷自动机（NFA）、NFA到DFA的确定化及DFA最小化等基础知识。深入理解词法分析过程和原理，能够初步设计和调试词法分析器，提升编译系统设计的实践能力。

## 二、实验内容与要求

根据给定的文法，设计词法单元的正规表达式及有穷自动机；

将多种单词符号的状态图合并，构造统一的NFA；

实现NFA到DFA的转换，并对DFA进行最小化；

利用状态转换矩阵，实现词法分析程序；

能够识别源程序中的关键字、标识符、常量、运算符、界符五大类单词；

识别结果以二元式形式输出，遇错误能提示位置并继续分析。

## 三、实验环境

操作系统：Windows 10

开发工具：Visual Studio Code

编译器：g++ 版本 9.4.0

语言：C++

## 四、实验原理与设计

### 4.1 词法单元的BNF定义

本实验词法单元采用如下BNF形式定义：

<标识符>→<字母>|<字母><字母数字>

<字母数字>→<字母><字母数字>|<数字><字母数字>|\_<字母数字>|ε

<字母>→a|b|...|z

<数字>→0|1|...|9

<常量>→<无符号整数>

<无符号整数>→<数字><数字串>

<数字串>→<数字><数字串>|ε

<运算符>→+|-|\*|/|=|<|==|<=|!=|>|>=

<关键字>→main|if|then|while|do

<界符>→{|}|(|)|;|,

### 4.2 有穷自动机设计

针对每个单词符号类型，设计对应的状态转换图（NFA），并进行合并形成统一的NFA。

**（插图1：标识符和数字NFA状态转换图）**

**（插图2：运算符及界符NFA状态转换图）**

### 4.3 NFA确定化与DFA最小化

实现NFA到DFA的确定化算法，采用子集构造法；随后使用Hopcroft算法对DFA进行最小化，降低状态数量。

**（插图3：NFA到DFA转换示意图）**

**（插图4：最小化后的DFA状态转换图）**

## 五、实验实现

### 5.1 代码结构介绍

main.cpp：程序入口，处理输入输出，调用词法分析模块。

lexer.cpp/h：核心词法分析模块，包含状态机构造、输入扫描和单词识别逻辑。

token.h：定义单词种别及属性结构。

### 5.2 关键代码说明

#### 5.2.1 状态转移矩阵定义

// 状态转移矩阵示例片段

int stateTable[MAX\_STATE][MAX\_INPUT\_CHAR];

// 初始化、状态转换逻辑

**（插图5：状态转移矩阵代码截图）**

#### 5.2.2 词法分析主循环

while (true) {

// 读取字符，状态转移

// 识别单词，输出二元式

// 处理错误，跳过非法字符

}

**（插图6：词法分析核心循环代码截图）**

## 六、实验运行结果

### 6.1 输入源程序示例

main()

{

int a, b;

a = 10;

b = a + 20;

}

#

### 6.2 终端输出截图

**（插图7：程序运行界面及终端输出截图，包括识别结果）**

### 6.3 生成的输出文件截图

**（插图8：output.txt 文件内容截图）**

### 6.4 错误识别演示

输入包含非法字符时，程序能正确提示错误位置并跳过，继续分析。

**（插图9：错误提示界面截图）**

## 七、实验体会与总结

通过本次词法分析器的设计与实现，深刻体会到正规文法到自动机转换的具体过程，加深了对NFA、DFA及最小化算法的理解。实现中遇到状态合并、错误处理等问题，通过调试逐步完善，提升了编译原理课程的实践能力。

本实验也锻炼了C++编程技能，尤其是状态机的设计思路及程序结构安排。后续可以尝试扩展支持更多词法规则，或者结合语法分析进行编译器前端设计。

# 附件

实验代码源文件

测试用例输入输出文件

实验指导书扫描件

如果你需要，我可以帮你做具体的代码截图或示例插图提示。需要告诉我你代码哪个部分截图，或者给我看截图，我帮你写说明文字。