**编译原理**

班级：23计算机科学与计算机

姓名：盲灯 学号：

实验一 TINY语言的词法分析

1. 实验目的

构造tiny语言的词法分析器（扫描器），要求利用第三方的lex工具进行构造。实验结果：构造出的扫描器，能够读入教材样例中给出的tiny语言的示例代码，分解成token输出。

1. 实验设计

1.安装 Flex

在开始实验之前，需要在系统中安装 Flex 工具。安装之后并将Flex添加进环境变量，

在cmd命令行输入flex --version，输出版本信息即表示安装完成。

2.配置GCC编译器

在MinGW官网安装MinGW,安装 GCC（如通过 MinGW）后，需将其安装路径添加到系统的环境变量 PATH 中,保存后即可在命令提示符中直接使用 gcc 命令进行编译。

3.编写lex程序，使用flex将其编译为c程序，使用gcc将c程序编译为可执行文件（扫描器）。

输入：tiny语言的示例代码的文件名。

输出：分解的token。

针对输入：

**{ Sample program**

**in TINY language -**

**computes factorial**

**}**

**read x; { input an integer }**

**if 0 < x then { don't compute if x <= 0 }**

**fact := 1;**

**repeat**

**fact := fact \* x;**

**x := x - 1**

**until x = 0;**

**write fact { output factorial of x }**

**end**

(1)包含头文件与全局变量

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#ifdef \_WIN32

    #include <io.h>      *// Windows系统文件操作*

#else

    #include <unistd.h>  *// Linux/macOS文件操作*

#endif

int line = 1;  *// 行号初始值为1*

%}

(2)定义状态

%x COMMEN

(3)正则表达式与动作:

数字标识符：[0-9]+

匹配一个或多个连续的数字字符。

字符串标识符：[a-zA-Z]+

匹配一个或多个连续的字母字符。

关键字标识符：read|if|then|repeat|until|write|end

匹配 TINY 语言中的预定义关键字。

操作符标识符：:="|";"|"<"|">"|"="|"+"|"-"|"\*"|"/"|"%"|

匹配 TINY 语言中的各种操作符。

空白字符：[ \t]+

匹配空格和制表符，用于忽略这些字符。

注释内容：[^{}\n]\*

匹配注释中的普通字符（不包括 {、} 和换行符）。

(4)辅助函数

yywrap 函数

int yywrap() { return 1; }

main 函数

int main(int argc, char \*argv[]) {

    setlocale(LC\_ALL, "");  *// 设置本地化环境*

    if (argc != 2) {

        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

        exit(1);

    }

    FILE \*file = fopen(argv[1], "r");

    if (!file) {

        fprintf(stderr, "Error: Cannot open file %s\n", argv[1]);

        exit(1);

    }

    yyin = file;

    printf("line 1:\n");

    yylex();

    fclose(file);

    return 0;

1. 内容和步骤

1代码：

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

 #ifdef \_WIN32

        #include <io.h>      *// Windows 系统*

    #else

        #include <unistd.h>  *// Linux/macOS*

  #endif

int line = 1;  *// 初始行号为 1*

%}

%x COMMENT

%%

\n          { printf("\nline %d:\n", ++line); }  *// 每遇到换行符更新行号*

"{"         { BEGIN(COMMENT); printf("COMMENT BEGIN\n"); }

<COMMENT>[^{}\n]\*    { printf("%s", yytext); }  *// 打印注释内容（不含 {、} 和换行）*

<COMMENT>\n { printf("\n"); ++line; }           *// 处理注释中的换行符*

<COMMENT>"}" { BEGIN(INITIAL); printf("\nCOMMENT END\n"); }

"read"      { printf("READ "); }

"if"        { printf("IF "); }

"then"      { printf("THEN "); }

"repeat"    { printf("REPEAT "); }

"until"     { printf("UNTIL "); }

"write"     { printf("WRITE "); }

"end"       { printf("END "); }

":="        { printf("ASSIGN "); }

";"         { printf("SEMICOLON\n"); }

"<"         { printf("LESS\_THAN "); }

"<="        { printf("LESS\_EQUAL "); }

">"          { printf("GREATER\_THAN "); }

"="         { printf("EQUALS "); }

"+"         { printf("PLUS "); }

"-"         { printf("MINUS "); }

"\*"         { printf("MULTIPLY "); }

"/"         { printf("DIVIDE "); }

"%"         { printf("MODULO "); }

[0-9]+      { printf("[NUMBER, Value=%s] ", yytext); }

[a-zA-Z]+   { printf("[IDENTIFIER, Name=%s] ", yytext); }

[ \t]+      { */\* 忽略空格和制表符 \*/* }

.           { printf("[ERROR: Unknown symbol '%s'] ", yytext); }

%%

int yywrap() { return 1; }

int main(int argc, char \*argv[])

{

    setlocale(LC\_ALL, "");  *// 设置本地化环境*

    if (argc != 2) {

        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

        exit(1);

    }

    FILE \*file = fopen(argv[1], "r");

    if (!file) {

        fprintf(stderr, "Error: Cannot open file %s\n", argv[1]);

        exit(1);

    }

    yyin = file;

    printf("line 1:\n");

    yylex();

    fclose(file);

return 0;

}

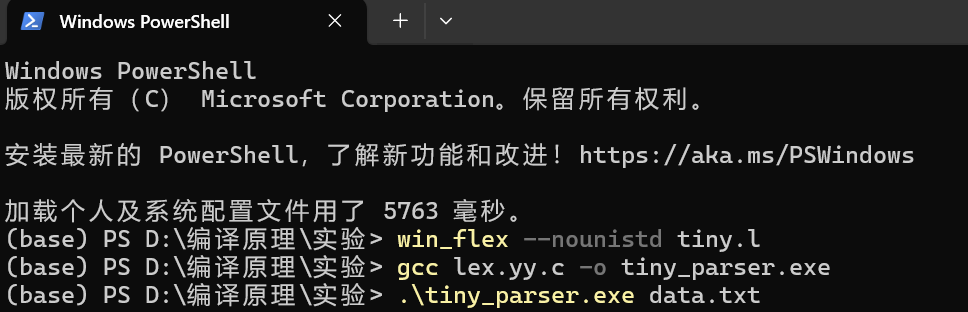
2、结果：

在tiny.l和data.txt的所在文件里打开cmd,然后依次编译命令：

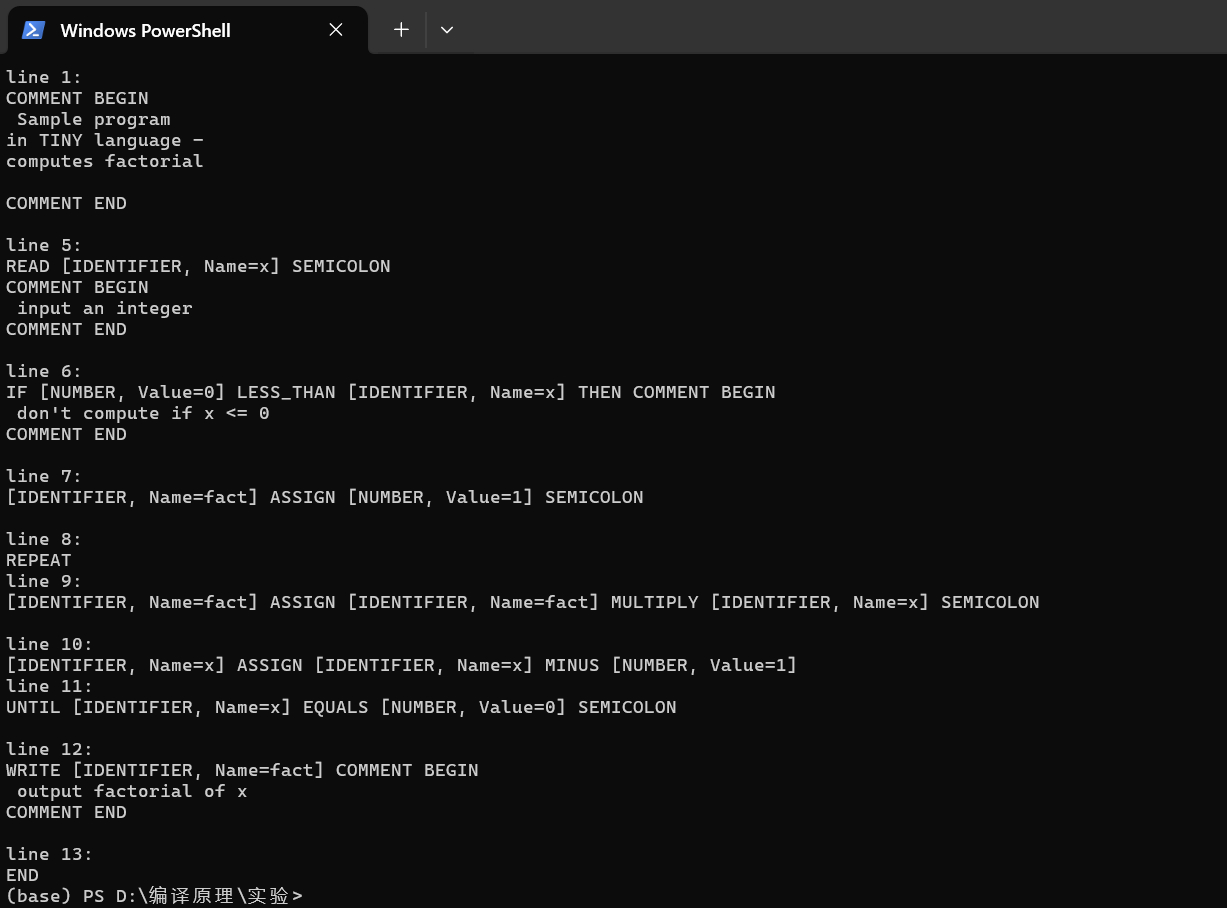
win\_flex --nounistd tiny.l

gcc lex.yy.c -o tiny\_parser.exe

.\tiny\_parser.exe data.txt



结果如图所示：



1. 实验结论:

1 理论基础：

（1）我掌握了词法分析的核心原理，通过实验，我理解了词法分析是编译的第一步，核心任务是将源代码拆解为最小语法单元，也就是Token。例如，通过定义正则表达式规则，我学会了如何让程序识别TINY语言中的关键字，如if、read、操作符，如:=、+和标识符，并输出对应的Token类型。

（2）我理解了正则表达式与自动机的联系，实验中，我通过编写[0-9]+匹配数字、<COMMENT>处理注释等规则，体会到Flex如何将正则表达式转换为有限状态自动机（DFA）。这让我明白，看似复杂的字符匹配逻辑，本质上是状态机的状态切换过程。

（3）我熟悉了编译工具链的协作流程，从编写Flex里的tiny.l文件到生成可执行程序，我实践了Flex和GCC的完整工具链：先用win\_flex生成C代码，再用GCC编译为可执行文件。这让我对“工具自动化生成代码”的编译思想有了抽象认识。

1. 分析和总结：

（1）我成功实现了TINY词法分析器，能够正确解析教材示例代码。例如：

注释部分被完整识别，COMMENT BEGIN和COMMENT END；

变量x被标记为[IDENTIFIER, Name=x]，数字0被标记为[NUMBER, Value=0]；

遇到非法符号（如$）时，程序会提示[ERROR: Unknown symbol]，体现了鲁棒性。

（2）有时需要注释跨行处理：当注释内容包含换行时，程序能正确更新行号并保持COMMENT状态，例如样例中的多行注释；以及操作符优先级问题：最初未正确处理<=，导致被拆分为<和=，通过调整正则表达式顺序解决了冲突；还有行号同步机制：换行符\n触发行号递增，帮助我在调试时快速定位错误位置。

（3）通过实验，我不仅学会了使用Flex工具，还锻炼了以下能力：

通过打印中间结果，如注释内容，验证状态切换是否正确；

意识到正则表达式的顺序会影响匹配优先级，如<和<=需严格排序；

将复杂的词法分析任务分解为关键字、操作符、注释等模块逐一实现。

1. 对工具的评价

（1）Flex的优势

高效便捷：仅需定义规则文件，Flex就能自动生成高效的词法分析代码，大大节省了开发时间；

跨平台支持：通过条件编译（如#ifdef \_WIN32），同一份代码可在Windows和Linux系统运行；

灵活的状态管理：利用%x COMMENT实现注释的独占状态，避免了普通字符与注释的冲突。

（2）Flex的不足

学习成本较高：初学时对“起始条件”“状态切换”等概念理解困难，需反复查阅文档；

错误提示不友好：若正则表达式存在歧义（如未正确处理<=），Flex不会主动报错，只能通过测试发现；

依赖外部工具链：必须配合GCC编译生成可执行文件，增加了环境配置的复杂度。

（3）局限性反思

无法处理复杂语法：仅适合词法分析，若需实现语法分析，如检查if后是否有then，必须结合Bison等工具；

国际化支持弱：默认不支持中文等非ASCII字符，若需扩展需手动修改编码规则；

性能瓶颈：虽然对小规模代码解析高效，但面对超长字符串（如未闭合的注释）时可能因状态堆积导致效率下降。