北京郵電大學

Beijing University of Posts and Telecommunications

北京邮电大学

计算机学院 (国家示范性软件学院)

C 语言词法分析程序设计

Author:

2019114514 班　田所浩二　2019114514

Course:
编译原理与技术

Supervisor:
刘辰副教授

2022 年 5 月 17 日

目录

1 需求分析

1.1 开发环境

本程序在 MacOS(类 Unix) 环境下开发, 使用 C 语言编写，使用 clang 编译器编译.

在方法二中，使用 flex 程序通过 lex.l 文件生成 lex.yy.c 代码

程序可以在各平台下编译运行。

1.2 功能及数据说明

本程序用于 C 语言代码的词法分析, 程序会读取代码文件, 对其进行词法分析, 并输出符号流, 以配合后续的语法分析/语义分析等。如遇到错误，会进行错误位置提示以及后续的错误恢复。

程序的输入数据为 C 语言源文件代码，输出为符号结构体流，符号结构体中包含符号类型，符号属性，以及符号原始值。

1.3 数据流图

程序从标准输入中读取代码, 将识别到的符号打印到标准输出中. 可以使用输入/输出重定向符号将输入/输出重定向至文件中, 使用以下命令运行本程序可以对 main.c 进行词法分析, 并将符号输出至标准输入流中:

```
$ ./lex_main < main.c # 使用由lex生成的lexer
$ ./mylexer_main < main.c # 使用自行编写的lexer
```

本词法分析程序的数据流图见图1

2 总体设计

2.1 数据结构设计

```
struct symbol {

    //符号类型
    symtype type;

    // 符号属性
    symattr attr;

    // 符号原始值
    const char* value;
}
```

代码块 1: symbol 符号结构体

```
struct wc {
    int lc;
    int bc;
    int comment;
    int keyword;
    int punc;
    int num;
    int literal ;
    int id;
    int null;
};
```

代码块 2: 符号个数结构体

本程序最重要的数据结构为符号数据结构`struct symbol`，其中包含符号的类型，符号属性以及符号原始值。用于统计文件各类符号个数/代码行数的结构体`struct wc`在程序中以全局变量的形式存在。

2.2 总体结构设计

2.2.1 功能模块划分

词法分析模块：提供词法分析函数。词法分析函数从标准输入流中读取 C 语言代码，返回识别到的符号类型。并统计代码行数。

词法分析模块使用两种方式实现:

- 使用 FLEX 程序，通过 lex.l 生成词法分析模块
- 自己使用 C 语言编写，通过提供与 lex 程序相同的 yylex() 函数接口来保持上层一致性。

符号结构体包装模块：将符号类型，符号原始值包装成符号结构体返回，并统计各类符号的个数。

符号打印模块：将符号结构体格式化打印出来。

主程序模块：调用其他的模块完成词法分析并打印的完整功能。

2.2.2 模块间关系

模块间关系图见图2。

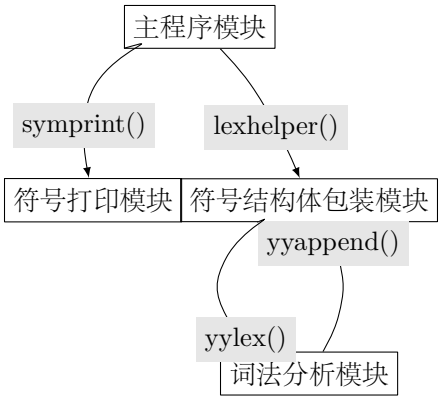

图 2: 模块关系图

2.2.3 模块间接口

语法分析模块

- `yylex()`: 提供词法分析功能，向前读入输入流直到识别到下一个符号或错误。返回符号类型或错误类型。

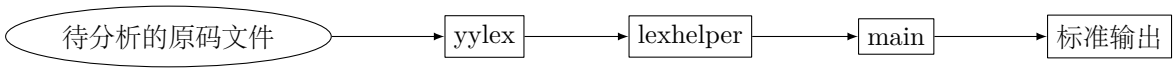

图 1: 数据流图

- `struct wc wc`: 提供全局变量，记录当前读取的字符总数、代码行数、各类符号个数等

符号结构体包装模块

- `lexhelper()`: 调用`yylex()`函数识别符号，并将识别到的符号类型与原始值打包至符号结构体中返回，最后递增 wc 全局变量中相应的符号计数器。
- `yystrinit() yyappend()`: 配合 DFA 进行字符串常量的识别。

符号打印模块

- `symprint()`: 格式化打印符号结构体。

2.3 用户接口设计

- `lexhelper()` 函数每次调用都返回下一个识别到的符号或错误，返回值类型为符号结构体：见代码块1
- `struct wc wc`: 提供全局变量，记录当前读取的字符总数、代码行数、各类符号个数等
- `symprint()`: 格式化打印符号结构体。

3 详细设计

3.1 词法分析函数

使用 LEX 程序生成的词法分析模块不再介绍，下面主要介绍 C 语言自行编写的词法分析函数。

其它接口的描述在前文已有叙述，这里着重介绍最核心的词法分析函数`yylex()`。词法分析模块需要实现三个重要操作：`Peek()`、`Get()`、`Eat()`，见图3。

在词法分析的过程中，为了确定下一个符号的类型，我们必须提前读取当前字符读头前的第 n 个字符。比如为了分辨符号 '+' 与 "++"，我们必须提前读取 1 个字符才能确定。所以我们有`peekch(n)`函数在不移动字符读头的条件下提前读取前面的第 n 个字符。

当确定了符号的类型后，我们想将长度为 n 的符号字符串从输入缓冲中拷贝至符号缓冲区中、并同时将读头前移 n 个字符，所以我们有`getch(n)`函数。

在遇到注释区、空白字符时，我们想要跳过它们而不将它们拷贝至符号缓冲区中、只是将读头前移 n 个字符，所以我们有`eatch(n)`函数。

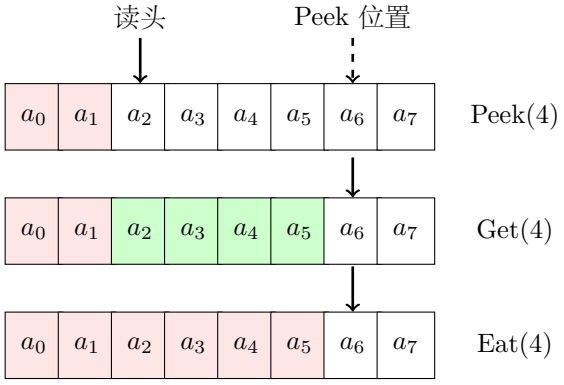

图 3: Peek(n), Get(n) 与 Eat(n)

这三个函数配合，就足以支撑词法分析程序需要的操作。

3.2 两段读入缓冲

设计两段文件缓冲的目的主要是要保障读头下的数据不会被新读入的数据意外覆盖。

如果我们只有一段缓冲，当读头在缓冲尾部，而实际读 (Peek) 的位置超出了缓冲区，我们就必须向缓冲区中读入新的数据，那么就有可能出现读头下的数据被新读入的数据意外覆盖的情况。

两段式文件缓冲则解决了这个问题。每当读头位置、或是 Peek 位置超出了当前缓冲区域时，我们便激活另一段缓冲区，向另一段缓冲区中读入数据。当读头离开旧缓冲区时，将读头放置到新缓冲区，并将旧缓冲区设置为未激活状态，此时两个缓冲区完成角色互换。

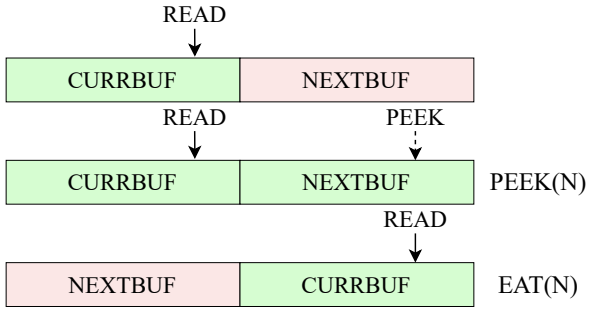

图 4: 缓冲角色互换

3.3 错误检测及恢复

在 C 语言词法分析的过程中，可能出现以下两种错误：

- 非法字符错误
- 词法规则错误

在有限状态自动机中，表现为状态在某个状态下接受一个字符，无法到达任意其他状态，此时词法分析就出现了错误。

错误恢复主要分为以下几个步骤：

- 将 DFA 的状态恢复至初态
- 判断错误类型
- 将读头移动至出错位置的下一个字符

在判断错误类型时，我们需要在设计自动机时，在自动机的状态转移表中加入错误类型。如果输入一个字符后状态转移表查询得到的内容是错误类型，说明发生了错误。我们可以首先将状态转移表的空项（无法接收该输入）的项全部设置为"非法字符"错误。再根据我们编程的经验，将某些错误项替换为有更好描述性的错误类型。

本程序支持的错误类型见表1

3.4 有限状态自动机设计

由于不同符号类型的识别过程不会互相转移，例如在一次符号识别的过程中若已处于识别字符串的状态，之后将不会再转移至识别数字常量的状态，所以可以简单地将整个词法分析模块的自动机分割成几个子自动机。本程序的所有子自动机如下：

- 词法分析总自动机
- 跳过注释自动机
- 跳过宏自动机
- 字符串识别自动机
- 数字常量识别自动机
- 字符常量识别自动机
- 关键字/标识符识别自动机
- 运算符识别自动机

词法分析总自动机的状态转移图见图5

4 程序测试

4.1 测试环境

本程序在 MacOS 系统下，使用 clang12 编译器编译运行。经过测试，本程序自身的源代码可以成功通过词法检测，但是由于篇幅限制，这里只贴出两个简单的测试用例。第一个测试用例用于测试程序在正确情况下的运行情况，第二个测试用例用于测试程序的错误提示与错误恢复能力。

4.2 测试功能

4.2.1 测试用例

```
//test
/* test
 * test
 */
#define TEST

long foo(short a) {
    return a + 0x1f2b3c;
}

int main() {
    const float asd = 123.20312;
    static char *assdsad = "asdsad\
                    asdsadsad";
    char ch = '\n';
    if (1) {} else {}
    switch(1) {case '1': default:;}
    for (;;) {break;}
    while(0) {continue;}
    int a = sizeof(double);
    struct asd;
    union c;
    return 0;
}
```

代码块 3: 测试样例 1

```
123.23
123.
0x123.213
0b00110
0b123
0x23213g
12321abc
"unterminated
'u
"newline with backslash\
newline"
@@@
/* unfinished block comment
```

代码块 4: 测试样例 2

错误类型	错误描述
ERR_UNTERMINATED_STRING_LITERAL	缺少了右双引号的字符串
ERR_UNTERMINATED_CHAR_LITERAL	缺少了右单引号的字符常量
ERR_UNTERMINATED_BLOCK_COMMENT	缺少右界的块注释
ERR_UNEXPECTED_IDENTIFIER	错误的标识符
ERR_NULL_CHARACTER_LITERAL	单引号中为空的字符常量
ERR_MAX_STRING_LENGTH_EXCEEDED	超出了 4096 字节最大长度限制的字符串
ERR_UNEXPECTED_CHARACTER	非法字符

表 1: 错误类型

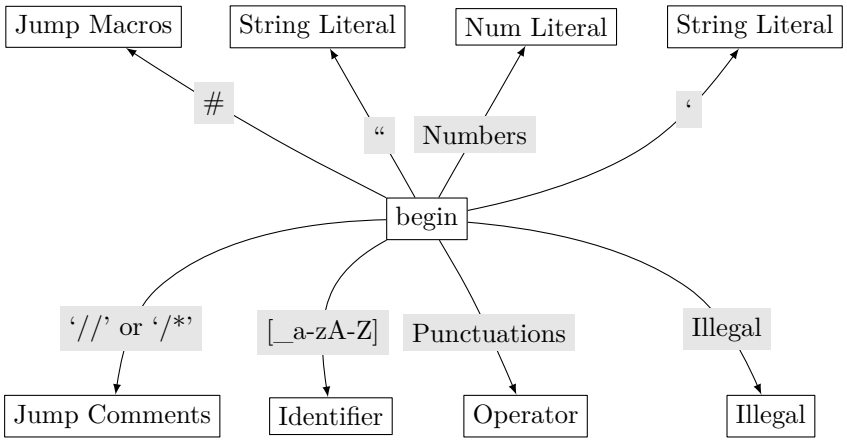

图 5: 词法分析总自动机

4.2.2 预期结果

针对正确的代码能够输出每一个符号的类型，属性及原始值。最后输出所有文件行数、字数以及各类型符号词数的统计。

针对有词法错误的代码能给出错误的行数、错误的类型以及错误的字符串。

4.2.3 测试结果及其分析

分析：针对测试样例 1 中的每一个关键字，程序都输出了一个代表该关键字的类型如 T_INT，T_DOUBLE 等。针对整数或浮点数字面值，则是由 T_NUM 代表其类型，由符号的属性提示符号属于 N 进制/浮点数类型。最后字符串字面值的类型名称为 T_LITERAL，单字符符号没有自定义的符号名称，而是使用它自己的单个字符作为符号名称。

针对测试样例 2 中的每一个错误，程序都输出了错误的位置、对应错误的类型与出错的字符串。

```
SYMTYPE SYMATTR SYMVALUE
macro
T_LONG ––– long
T_ID   ––– foo
(    ––– (
T_SHORT ––– short
T_ID   ––– a
)    ––– )
{    ––– {
T_RETURN ––– return
T_ID   ––– a
+    ––– +
T_NUM ATTR_NUM_HEX 0x1f2b3c
 ...
T_NUM ATTR_NUM_FLOAT 123.20312
 ...
T_LITERAL ––– ”asdsad\
                      sadsad\
                      asdsadsad\
                      asdsadsad”
;    ––– ;
T_CHAR ––– char
 ...
T_UNION ––– union
T_ID   ––– c
;    ––– ;
T_RETURN ––– return
T_NUM ATTR_NUM_DECIMAL 0
;    ––– ;
}    ––– }
代码行数：28
总字节数：396
注释数：2
关键字数：24
标点符号数：49
数字常量数：8
字符串常量数：1
标识符数：10
未识别符号数：0
```

代码块 5: 测试输出 1(有删减)

```
SYMTYPE SYMATTR SYMVALUE
T_NUM ATTR_NUM_FLOAT 123.23
T_NUM ATTR_NUM_FLOAT 123.
line  3: ERR_UNEXPECTED_IDENTIFIER 0x123
     .213
T_NUM ATTR_NUM_BIN 0b00110
line  5: ERR_UNEXPECTED_IDENTIFIER 0b123
line  6: ERR_UNEXPECTED_IDENTIFIER 0x23213g
line  7: ERR_UNEXPECTED_IDENTIFIER 12321abc
line  9:
     ERR_UNTERMINATED_STRING_LITERAL ”
     unterminated

line  9: ERR_UNTERMINATED_CHAR_LITERAL ’
     u
T_LITERAL ––– ”newline with backslash\
newline”
line  12: ERR_UNEXPECTED_CHARACTER @
line  12: ERR_UNEXPECTED_CHARACTER @
line  12: ERR_UNEXPECTED_CHARACTER @
line  14:
     ERR_UNTERMINATED_BLOCK_COMMENT
代码行数：13
总字节数：137
注释数：0
关键字数：0
标点符号数：0
数字常量数：3
字符串常量数：1
标识符数：0
未识别符号数：3
```

代码块 6: 测试输出 2

5 源程序清单

File Name	Description
symbol.h	定义符号数据结构, 包括符号类型枚举变量等
symbol.c	定义 symname, attrname 符号类型与符号名称对应表全局变量
lexprint.c	定义符号打印函数
lexhelper.c	将 yylex() 返回的符号类型值包装成符号结构体返回
main.c	主函数, 循环调用 yylex() 函数并打印符号, 进行错误处理
lex.l	用于方法二 LEX 程序生成代码的源文件
lexer.c	用于方法一, 手写的词法分析程序, 提供功能相似的 yylex 函数
Makefile	make 编译脚本
test/a.c	测试样例 1
test/err	测试样例 2
report/*	实验报告源码
实验报告.pdf	实验报告

表 2: 源程序清单