实现词法分析器

潘宇

September 2019

目录

1 作业要求 **3**

2 作业解析 **4**

2.1 作业目的 4

2.2 作业思路 4

2.3 Lex 程序结构讲解 4

2.4 设计正则定义，编写转换规则 5

2.5 符号表的实现 7

2.6 文件输入 8

1 作业要求

列出你的 C 编译器支持的语言特性所涉及的单词，设计正规定义。
利用 Lex 工具实现词法分析器，识别所有单词，能将源程序转化为单词流。
设计符号表，当然目前符号表项还只是词素等简单内容，但符号表的数据结构，搜索算法，词素的保存等等都可以考虑了。保留字的处理等问题也可考虑了。
验证你的程序，可以输入简单的 C 源程序，输出单词流每个单词的词素内容、单词类别和属性（常数的属性可以是数值，标识符可以是指向符号表的指针）。其中类似 PPT 代码，输入的 C 源程序请使用文件输入的方式。如下例：

```
1  main()
2  {
3          int a;
4          if (a == 0)
5                  a = a + 1;
6  }
```

可能的输出结果为：

```
1  单词              词素      属性
2  ID                main      0
3  LPAREN            (
4  RPAREN            )
5  LBRACE            {
6  INT               int
7  ID                a         1
8  SEMICOLON         ;
9  IF                if
10 LPAREN            (
11 ID                a         1
12 EQ                ==
13 NUMBER            0         0
14 RPAREN            )
15 ID                a         1
16 ASSIGN            =
17 ID                a         1
18 PLUS              +
19 NUMBER            1         1
20 SEMICOLON         ;
21 RBRACE            }
```

给老师/助教讲解你的程序、进行展示，并提交 Lex 程序

2 作业解析

2.1 作业目的

通过本次实验，希望同学们能够加强 Parser Generator 工具的掌握与使用。与上次实验所对应的 Yacc 程序，这次实验希望同学们能够学会 Lex 程序的编写，复习巩固上课所学与正规表达式相关的知识，加深对 Lex 程序作用以及词法分析器的工作原理及作用的理解，提高实践能力。

2.2 作业思路

根据作业要求，首先需要根据第二次实验所写的你的 C 编译器支持的语言特性，对其所涉及的单词设计正规定义，以便编写 lex 程序时使用。然后使用 Parser Generator 工具编写 lex 程序，通过正规定义识别所有单词，将源程序转化为单词流，之后在 lex 程序的规则段定义执行动作，用于实现符号表功能，输出单词流中每个单词的词素内容、单词类别和属性。其中常数的属性可以是数值，标识符可以是指向符号表的指针，而其他单词不必输出属性。同学们可以使用数组，vector，map 等数据结构实现符号表。对于主函数，需要将输入流由标准输入修改为文件输入，同学们可以参考第三章 PPT 的第 56 页和第 59 页。

2.3 Lex 程序结构讲解

使用 Parser Generator 工具生成的初始 Lex 程序如下：

```
%{
/**************************************************************************
mylexer.l
ParserWizard generated Lex file.

Date: 2019年9月22日
**************************************************************************/
%}

//////////////////////////////////////////////////////////////////////////
// declarations section

// lexical analyser name
%name mylexer

// class definition
{
        // place any extra class members here
}

// constructor
```

```
22 {
23         // place any extra initialisation code here
24 }
25
26 // destructor
27 {
28         // place any extra cleanup code here
29 }
30
31 // place any declarations here
32
33 %%
34
35 ////////////////////////////////////////////////////////////////////////
36 // rules section
37
38 // place your Lex rules here
39
40 %%
41
42 ////////////////////////////////////////////////////////////////////////
43 // programs section
44
45 int main(void)
46 {
47         int n = 1;
48         mylexer lexer;
49         if (lexer.yycreate()) {
50                 n = lexer.yylex();
51         }
52         return n;
53 }
```

其中 1-8 行用于添加所需的头文件和命名空间，注释部分可添加描述 lex 文件信息。declarations section 为声明部分，其中第 13、14 行命名词法分析器名称，用户子程序段的 main 函数的第 48 行需要与其对应，第 16-19 行用于声明额外的各种变量和类，第 21-29 行对应额外的初始化操作代码和 clear 回收操作代码，第 31-32 行用于声明正规定义和其他额外声明。rules section 部分，为规则段，用于编写转换规则，转换规则包括模式和动作两个部分，其中每个模式是一个正则表达式，它也可以使用声明部分中给出的正则定义，动作部分为代码片段，转换规则可添加在第 39 行的位置，programs section 为用户子程序段，用于编写用户子程序，其中 45-53 行为 main 函数。

2.4 设计正则定义，编写转换规则

在上一小节已经介绍过转换规则，转换规则的模式部分既可以是正则表达式，也可以是声明部分给出的正规定义。因此如果只为实现本次作业程序功能的话，既可以正在声明部分对每个单词设计正规定义，然后在规则段

编写转换规则时模式使用设计的正规定义，也可以不进行正规定义，在编写转换规则时直接使用正则表达式。但是为了增强代码的可读性以及提高同学们对正规定义的理解，并且设计正规定义是本次作业要求之一，所以**本次作业需要同学们采用在声明部分对每个单词设计正规定义，然后在规则段编写转换规则时模式使用设计的正规定义的方法**。不过为了同学们掌握这两种情况，接下来我将对应这两种情况各举例子。

在举例子之前，需要提醒同学们编写 lex 代码的正则表达式时需要注意正则表达式的扩展以及特殊字符问题，可参考龙书 P78 的 3.3.5 节和 P79 的图 3-8 以及第三章 PPT 的 52 页。

以分号为例，介绍转换规则的模式部分直接使用正则表达式的情况。很容易知道分号的正则表达式就是分号。因此模式部分为分号";"，动作部分首先输出单词 SEMICOLON 然后输出词素分号";"，代码如下：

```
...
// rules section

// place your Lex rules here
";"             {cout << "SEMICOLON\t" << yytext << endl;}

%%
...
```

其中 yytext 为字符串类型，代表单词流的单词所对应的字符串字面值。

然后以加号为例，说明在声明部分对每个单词设计正规定义，然后在规则段编写转换规则时模式使用设计的正规定义的方法。在声明部分给加号正则表达式命名为 PLUS，需要注意加号为特殊字符，写正则表达式时不要忘记转义字符，然后在编写加号的转换规则时模式部分使用正规定义 PLUS，动作部分首先输出单词 PLUS 然后输出词素符号"+"，代码如下：

```
...
// place any declarations here
PLUS \+
%%

////////////////////////////////////////////////////////////////////////////
// rules section

// place your Lex rules here
{PLUS}          {cout << "PLUS\t\t" << yytext << endl;}
";"             {cout << "SEMICOLON\t" << yytext << endl;}

%%
...
```

其余单词的正规定义的设计及转换规则的编写请同学们举一反三实现。同学们涉及的单词要尽可能全面，为下一次实验作铺垫。

2.5 符号表的实现

符号表可以用很多数据结构实现，如数组、vector、map 等等。这里我使用 unordered_map 实现。针对作业中的要求，只需要实现一个符号表保存出现的标识符，并对每一个标识符给予一个指针位置（用数字表示），对于符号表的维护更新，当出现一个标识符后，遍历符号表的符号表项，若已存在，则输出对应的指针位置，若不存在，则表示该标识符第一次出现，将其加入符号表中。按照上述思路，需要一个 unordered_map 对象，key 为 string 类型，value 为 int 类型，还需要一个整数型变量 idcount 用于记录新的标识符的指针位置应赋多少，两个变量可声明在 class definition 处，代码实现如下：

```
%{
...
#include <fstream>
#include <iostream>
#include <string>
#include <unordered_map>
%}
...
// class definition
{
        // place any extra class members here
        unordered_map<string, int> idlist;
        int idcount = 0;
}
...
// place any declarations here
ID [A-Za-z_][A-Za-z0-9_]*
PLUS \+
%%

//////////////////////////////////////////////////////////////////////////
// rules section

// place your Lex rules here
{ID}            {
                    if(idlist.find(yytext) == idlist.end())
                    {
                        idlist[yytext] = idcount;
                        idcount++;
                    }
                    cout << "ID\t\t" << yytext << "\t" <<idlist[yytext]<< endl;
                }
{PLUS}          {cout << "PLUS\t\t" << yytext << endl;}
";"             {cout << "SEMICOLON\t" << yytext << endl;}

%%
...
```

同学们可以选用其他数据结构实现，另外目前的符号表实现较为简单，只为实现作业功能，鼓励学有余力的同学利用指针位置和其他数据结构实现更为复杂功能全面的符号表，不做硬性要求。

2.6 文件输入

只需将 lexer 的成员 yyin 赋值为文件输入流即可，代码如下：

```
1  int main(void)
2  {
3     int n = 1;
4     mylexer lexer;
5     if (lexer.yycreate()) {
6           lexer.yyin = new std::ifstream("a.txt");
7           cout << "单词\t\t词素\t属性" << endl;
8        if (!lexer.yyin->fail()) {
9           n = lexer.yylex();
10          }
11    }
12    system("pause");
13    return n;
14 }
```