**编译原理实验 Lab1-1190201303-王艺丹**

**实验完成情况综述**

完成所有必做内容以及选做所有内容：测试用例test1-test10均通过

**完成C--语言文法的词法分析：**即识别所有合法词法单元（包括选做任务的单多行注释），识别特定非法词法单元（包括非选做任务的八进制数、十六进制数、科学计数法表示的浮点数，以及错误以数字开头的标识符），以及未在C--文法中定义的词素（如&、！等）

**完成C--语言文法的所有语法分析：**即归约所有合法的语法单元，识别特定的非法语法（如分号缺失、大括号缺失），以及识别概括性的特定语法错误（如无效的Extern Definition等）；最重要的，在if-then-else语句中增加了特定语法以消除二义性；完成语法分析树的创建，完成语法树的正确输出

**实验编译方法**

**文件组成结构**

└── Lab1-1190201303-王艺丹

├── code ├── lexical.l ├── main.c

├── makefile ├── syntax.y ├── TreeNode.c └── TreeNode.h

├──test

├── test1.cmm ├── ……

└── test10.cmm

└── Lab1-Report.pdf

└── README.md

**编译方法**

在code目录下使用make命令编译所有文件

在code目录下使用make test命令进行词法语法分析，对测试样例进行测试

**实验完成功能及亮点**

**词法分析**

在文件中:

* 完成了C--语言文法所有基础合法词法单元的识别，调用函数存入终结符节点信息，

并将该节点赋值给语法单元，向语法分析器返回该词法单元的名

* 将特点非法词法单元识别为词法错误，分别编写十进制、八进制、十六进制整型正确以及错

误的正则表达式，使得词法任务分析时可以具体输出词法错误单元的类型，如：

* 将0x3F识别为
* 将09识别为
* 将1.05e识别为
* 将非法字符识别为

相关正则表达式：

1. INT 0|([1-9][0-9]\*)
2. DECERROR [0-9]+[a-wA-Wy-zY-Z]+[0-9a-dA-Df-zF-Z]*\*|[0-9]+[0-9]+[a-dA-Df-zF-Z]+[0-9]\**[0-9a-dA-Df-zF-Z]\*
3. OCT 0[0-7]+
4. OCTERROR 0[0-7]*\*[8-9]+[0-9]\**
5. HEX 0[xX][0-9a-fA-F]+
6. HEXERROR 0[xX][0-9a-fA-F]*\*[g-zG-Z]+[0-9a-zA-Z]\**
7. FLOAT   [0-9]+\.[0-9]+([eE][+-]?[0-9]+)?|[0-9]+[eE][+-]?[0-9]+|\.[0-9]+|[0-9]+\.|[eE][+-]?[0-9]+|[.][0-9]+[Ee][+-]?[0-9]+|[0-9]+\.[Ee][+-]?[0-9]+
8. FLOATERROR \.[eE][+-]?[0-9]+|[0-9]*\*\.[0-9]+[eE][+-]?|[0-9]+\.[0-9]\**[eE][+-]?|[0-9]+[Ee][+-]?|\.[eE][+-]?

* **附加功能中注释/\*\*/的识别处理**

规则编写如下：**必须读入连续的’/\*’与’\*/’**

1. "/\*"    {
2. char a = input();
3. char b = input(); //必须是连续的'/\*','\*/'
4. while(a != '\*' || b != '/') {
5. a = b;
6. b = input();
7. if(b == EOF){
8. printf("Error type B at Line %d: Missing \*/.\n",yylineno);
9. break;}
10. }
11. }

* 如果我们识别到了特定的非法词法单元，就将该词法单元的赋值为，向语法分析器返回或者两种自定义名，并在语法分析器里将这两个词法单元识别为合

法的语法单元，**避免了将词法错误传递为语法错误**；而当识别到空白符或未定义非法字符，

就不向语法分析器返回值，从而直接跳过这些字符，以边不影响接下来的语法分析

**语法分析**

在文件中:

* 完成了C--语言文法所有基础合法语法单元的识别，调用函数存入非结点信息，

并将该产生式的各产生体与该结点连接起来，将结点(即开始结点)作为整个语法生成

树的根节点

* 完成了C--语言文法全部非法语法单元的识别，并且将特定非法语法单元标识出来，如：
* 添加了特定归约，使得识别出语法单元组成中缺失’;’(如全局变量、结构体的声明、

局部变量的定义及表达式)

* 添加了特定归约，使得识别出语法单元组成中缺失’}’(如结构体的声明、函数定义)
* 自定义了与两种名以及产生式，使得语法分析中越过词法分

析错误，不至于重复报错(只报告词法错误)

**语法树建立**

在文件中:

树节点结构定义：

1. typedef enum Type
2. {
3. terminal\_int,
4. terminal\_hex,
5. terminal\_oct,
6. terminal\_float,
7. terminal\_id,
8. terminal\_type,
9. terminal\_other,
10. non\_terminal
11. }Type;
12. typedef struct TreeNode
13. {
14. int lineno; *// line number of the lexical unit*
15. Type type; *// type of the lexical unit*
16. char\* value;    *// value of the lexical unit(yytext)*
17. struct TreeNode\* firstChild, \*nextSibling;
18. }TreeNode;

在文件中:

**创建节点函数**

**该函数的功能为：创建一个行号为，类型为，需要保存的值为，其各孩子**

**结点数为的不定数量的，孩子结点指针在后面给出的结点（其中时对应终结符）**

代码如下：

1. pNode createNode(int \_lineno, Type \_type, char\* \_value, int args, ...)
2. {
3. pNode currNode = (pNode)malloc(sizeof(TreeNode));
4. assert(currNode != NULL);
5. currNode->value = (char\*)malloc(sizeof(char) \* (strlen(\_value) + 1));
6. assert(currNode->value != NULL);
8. currNode->firstChild = NULL; *//左孩子*
9. currNode->nextSibling = NULL; *//右兄弟*
10. currNode->lineno = \_lineno;
11. currNode->type = \_type;
12. strncpy(currNode->value, \_value, strlen(\_value)+1);
14. if(args > 0) *// terminal 无孩子兄弟节点*
15. {
16. va\_list ap;
17. va\_start(ap, args);
18. pNode tempNode = va\_arg(ap, pNode);
19. currNode->firstChild = tempNode;
20. for (int i = 1; i < args; i++){
21. tempNode->nextSibling = va\_arg(ap, pNode);
22. if(tempNode->nextSibling != NULL)
23. tempNode = tempNode->nextSibling;
24. }
25. va\_end(ap);
26. }
27. return currNode;
28. }

使用了库中提供的**类，通过、、函数用于解决变**

**参问题**，**并且在每次不同的归约产生式生成非终结符结点或终结符结点时，仅通过一次函数调**

**用便完成想要实现的功能。**

**输出打印语法树函数**

通过递归先序遍历左子树实现语法树打印（对应左孩子右兄弟）

代码不做赘述