编译原理

实

验

报

告

**实验名称：实验二：自底向上的语法分析LR(1)**

**姓名： 方澳阳**

**学号： 180110115**

**学院： 计算机科学与技术**

**专业： 计算机类**

# 实验目的与方法

（1）深入了解语法分析程序实现原理及方法

（2）理解LR(1)分析法是严格的从左向右扫描和自底向上的语法分析方法

实现语言: C++

环境: Clion2020.2， linux+windows

(如果运行有问题请在linux环境下运行)

# 实验总体流程与函数功能描述

**流程：**

1. 自行设计文法，定义一种类C语言的文法
2. 将由软件生成的LR(1)分析表读入程序中
3. 根据LR(1)的推导规约过程编写程序，输出推导过程中所用的产生式序列并保存在producer.txt中

# 实验内容

1. 有穷自动机的图片过大， 请到文件夹内自行查看。
2. LR(1)分析表也请到文件夹内自行查看。

### 数据结构设计

这里先给出词法分析时的数据结构设计。

#### Token串和符号表的逻辑结构及存贮结构

Token串的存储使用一个动态数组存储，定义为

vector<pair<int，string>>

数组中的每一个元素为一个pair，pair.first代表种类，pair.second表示具体的串。当pair.first为标识符时，可以通过pair.second去符号表中索引到对应的信息。

符号表使用Map作为存储结构。通过设置对应的string串作为key，可以在Map中得到其对应的value。符号表定义为

map<string，SYSTABLE> sysMap

其中，SYSTABLE为一个类，里面存储了对应标识符的type和address。

为了使词法分析和语法分析的接口对应，因此语法分析的输入数据的数据结构与token串相同，即为vector<pair<int，string>>。

#### LR分析表数据结构设计

由于LR（1）分析表可以分为两个表，即Action表和Goto表，因此我将两个表分开存储。

观察csv文件中的分析表可知，action表中的值的类型为字符串，而goto表中的为数字，因此声明部分为：

vector<vector<string>> actionMap;

vector<vector<int>> gotoMap;

由于C++没有像python一样的可以处理frame类型的数据，因此我将action表和goto表的索引单独存放，可以将字符串类型映射为数字类型，从而在上面的二维数组中找到对应的值：

map<string， int> index;

map<string， int> NoEndIndex;

## 算法描述

LR(1)分析算法伪代码为:

输入：文法G的LR分析表和输入串input

输出：如果input属于L(G)，则输出input的自底向上分析，否则报错。

步骤：

1. 将$和初始状态0压入栈，将input$放入输入缓冲区
2. 令输入指针pointer指向input$的第一个符号
3. 令S为栈顶状态，a是pointer所指向的符号
4. repeat
5. if action[S，a] = Si then
6. begin
7. 把符号a和状态i分别压进符号栈和状态栈
8. 令pointer指向下一个符号
9. end
10. else if action[S，a] = rk then
11. begin
12. 从符号栈和状态栈分别弹出|β|个符号
13. 令S’是现在的栈顶状态
14. 把A和goto[S’，A]分别压入符号栈和状态栈
15. 输出产生式A->β
16. end
17. else if action[S，a] == acc then
18. return
19. else
20. error();

#### 设计技巧

在实现上述伪码时，通过设计一些解析函数以及数据结构可以使分析过程更为简单。如分析表中的string值为reduce A->B - C，则通过parseState和parseProducer函数将该string解析为三个部分，即reduce，A，B-C， 分别表示规约、产生式头部、产生式尾部，从而实现伪码的功能

# 实验结果与分析

输入代码源程序为：

int a #

a = 2 #

int b = 3#

int c#

int fuc(a,b,c){

int d=0#

for(int i = 2, i<10,i++){

d=d+1

}#

c=d++

}#

fuc(a,b,c)#

b = c/2

通过词法分析产生的token串为

(17,int)

(85,a)

(80,#)

(85,a)

(72,=)

(82,2)

(80,#)

(17,int)

(85,b)

(72,=)

(82,3)

(80,#)

(17,int)

(85,c)

(80,#)

(17,int)

(85,fuc)

(44,()

(85,a)

(48,,)

(85,b)

(48,,)

(85,c)

(45,))

(59,{)

(17,int)

(85,d)

(72,=)

(83,0)

(80,#)

(14,for)

(44,()

(17,int)

(85,i)

(72,=)

(82,2)

(48,,)

(85,i)

(68,<)

(82,10)

(48,,)

(85,i)

(66,++)

(45,))

(59,{)

(85,d)

(72,=)

(85,d)

(65,+)

(82,1)

(63,})

(80,#)

(85,c)

(72,=)

(85,d)

(66,++)

(63,})

(80,#)

(85,fuc)

(44,()

(85,a)

(48,,)

(85,b)

(48,,)

(85,c)

(45,))

(80,#)

(85,b)

(72,=)

(85,c)

(50,/)

(82,2)

再经过语法分析，输出的产生式为：

type -> int

S' -> type token

C -> S'

B -> constV

A -> B

S -> A

S' -> token = S

C -> C # S'

type -> int

B -> constV

A -> B

S -> A

S' -> type token = S

C -> C # S'

type -> int

S' -> type token

C -> C # S'

type -> int

list -> token

list -> token , list

list -> token , list

type -> int

B -> constV

A -> B

S -> A

S' -> type token = S

C -> S'

type -> int

B -> constV

A -> B

S -> A

S' -> type token = S

B -> token

A -> B

S -> A

comOp -> <

B -> constV

A -> B

S -> S comOp A

B -> token

A -> B

A -> A ++

B -> token

A -> B

S -> A

B -> constV

A -> B

S -> S + A

S' -> token = S

C -> S'

S' -> for ( S' , S , A ) { C }

C -> C # S'

B -> token

A -> B

A -> A ++

S -> A

S' -> token = S

C -> C # S'

S' -> type token ( list ) { C }

C -> C # S'

list -> token

list -> token , list

list -> token , list

S' -> token ( list )

C -> C # S'

B -> token

A -> B

B -> constV

A -> A / B

S -> A

S' -> token = S

C -> C # S'

# 实验中遇到的困难与解决办法

和实验一一样，一开始还是不清楚本次实验需要做什么。课上老师说当做分析表已经有了，先把LR1的算法源码实现出来，然后一步一步向前推。当时课上完成了源码实现以及分析表的读入和存储。后来由于考试原因，接近10天没有碰该实验。导致几天前开始做的时候根本想不起来自己之前做了啥，为什么这么做。

在搞清楚要做什么之后，发现当时写的代码并不符合用软件生成的LR1分析表的格式，因此又花了很多时间去调整接口，重构代码。

可以说该实验分成了四个阶段。第一个阶段是不清楚自己要做什么，按照书上源码实现了算法；第二阶段是逐渐了解到自己要做什么，并且设计了简单的文法，在软件上测试之后，尝试读入分析表，但是会有很多错误，如漏掉了最后一个数据；第三阶段是分析表读入没有问题了，但是自己写的LR1算法出现了很多bug，毕竟从理论到实际还是有较大的的差距的；最后一个阶段就是算法没有问题了，需要设计更加完善的文法。

感到有些遗憾的是词法分析自己完成了很多功能，语法分析只实现了一小部分，再后面可能还需要继续阉割，这种逐渐失去的过程是比较难受的。并且自己设计的文法有着许多的问题，没有遵从规范，例如for循环中可以定义函数。这距离真正的编译器的功能还有很大的距离。