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

一、 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)

二、 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

三、 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

四、 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'

五、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。