同濟大學

TONGJI UNIVERSITY

§语法分析器设计——算符优先分析算法

班级	计算机科学 1 班
学号	1551265
姓名	张伯阳
时间	2017 年 11 月 27 日

实验内容与要求

1. 根据算符优先分析算法，编写一个语法分析程序，可以选择以下三项之一作为分析算法中的输入：

1) 直接输入根据已知文法人工构造的算符优先关系矩阵

2) 输入已知文法和 FIRST、LAST 集合，由程序自动生成该文法的算法优先关系矩阵

3) 输入已知文法，由程序自动生成该文法的算法优先关系矩阵

2. 程序具有通用性，即所编制的语法分析程序能够适用于不同文法以及各种输入单词串，并能够判断该文法是否为算符文法和算符优先文法。

3. 对输入的一个文法和一个单词串，程序能正确判断此单词是否为该文法的句子，并要求输出分析过程和语法树。

实验内容选择

1. 选择第三项要求:

输入已知文法，由程序自动生成该文法的算法优先关系矩阵

2. 程序具有通用性，即所编制的语法分析程序能够适用于不同文法以及各种输入单词串，并能够判断该文法是否为算符文法和算符优先文法。

3. 对输入的一个文法和一个单词串，程序能正确判断此单词是否为该文法的句子，并要求输出分析过程和语法树。

实验环境

开发语言:C++

开发环境:Mac OSX 平台下的 g++编译器

版本:Apple LLVM 9.0.0

由于生成的可执行文件为 Unix 可执行文件 故在 Windows 环境下无法运行

算符优先分析法

算符优先文法

定义：设有不含空串的一文法 G，如果 G 中没有形如 G>……BC……的产生式，其中 B 和 C 为非终结符，且对任意两个终结符 a，b 之间之多只有<，>，=，三种关系的一种成立，则称 G 是一个算符优先文法。

非终结符的 FIRSTVT 集合和 LASTVT 集合

FIRSTVT（B）={b|B→b…或 B→Cb…}

LASTVT（B）={b|B→…a 或 B→…aC}

算符优先矩阵

算符优先关系矩阵，判断输入是否满足已知文法的依据。根据非终结符的 FIRSTVT 集合和 LASTVT 集合产生。

1.“=”关系

若 A→…ab…或 A→…aBb…，则 a=b；

2.“⟨·”关系

若 A→…aB…，对每一个 b 属于 FIRSTVT（B），有 a⟨·b；

3.“·⟩”关系

若 A→…Bb…，对每一个 a 属于 LASTVT（B），有 a·⟩ b。

如何规约

在分析过程中，利用分析栈存放已识别的那部分句型，而句型的其余部分由剩余输入串组成，通过比较栈顶符号和下一个输入符号之间的关系，可以判别栈顶符号是否为句柄尾符号。如果是句柄尾，则沿栈顶向下，在栈内寻找句柄头。由之间包括的符号串就是句柄，然后把它们弹出栈，并代之以归约后的非终结符。这样就完成了一次归约过程。

算符优先分析方法的局限性

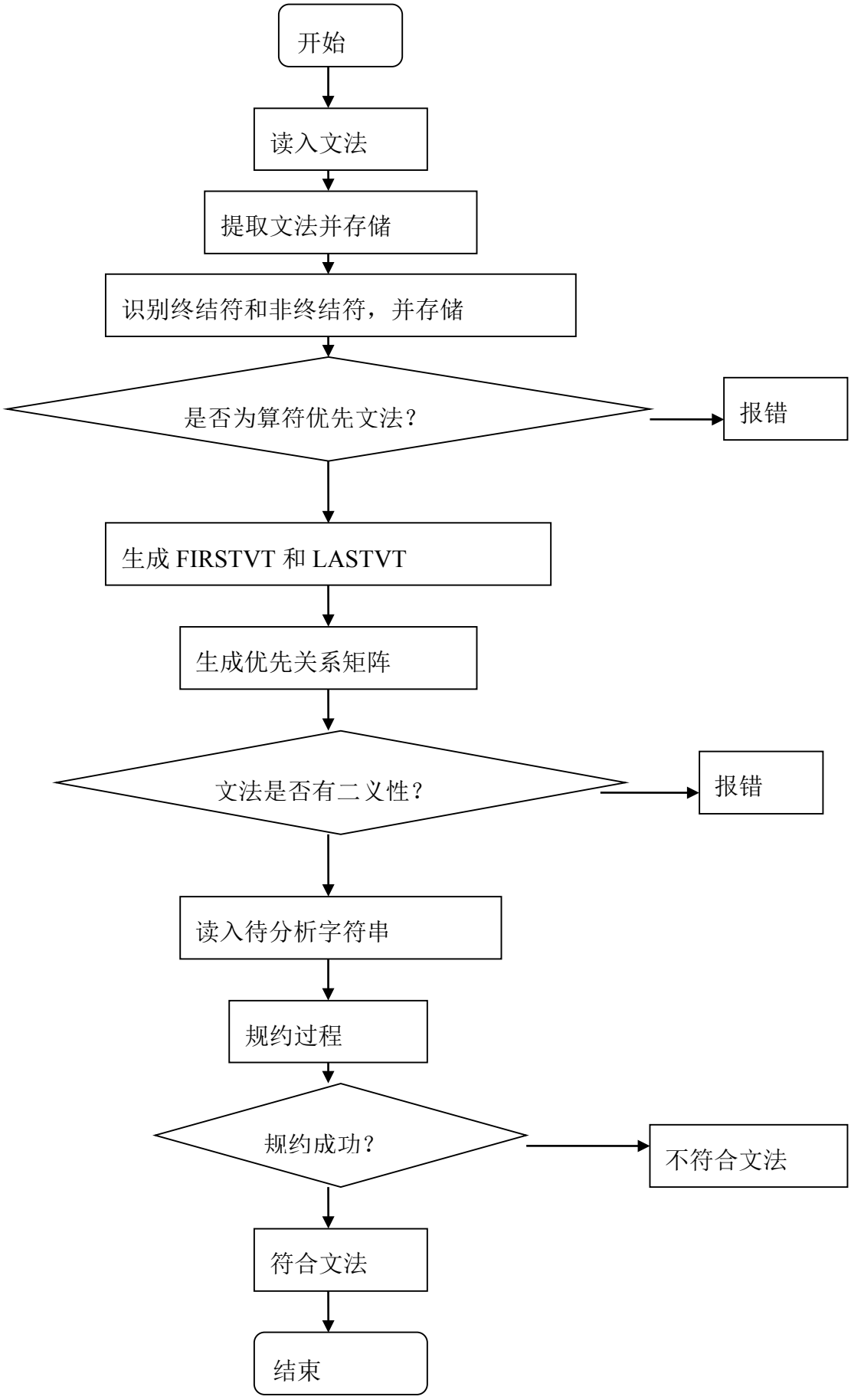
由于算符优先分析法去掉了单非终结符之间的归约，尽管在分析过程中，当决定是否为句柄时采取一些检查措施，但仍难完全避免把错误的句子得到正确的归约。

此外，通常一个适用语言的文法也很难满足算符优先文法的条件，因而致使算符优先分析法仅适用于表达式的语法分析。

概要设计

算符优先文法的执行过程为：输入已知文法，分析其正确性，提取非终结符和终结符，构造非终结符的 FIRSTVT 集和 LASTVT 集，再次基础上构造算符优先关系矩阵，并用来判断表达式是否符合该文法。

算符优先文法程序总的流程图为：


开始
读入文法
提取文法并存储
识别终结符和非终结符，并存储
是否为算符优先文法？
报错
生成 FIRSTVT 和 LASTVT
生成优先关系矩阵
文法是否有二义性？
报错
读入待分析字符串
规约过程
规约成功？
不符合文法
符合文法
结束

文法每个非终结符的 FIRSTVT 集和 LASTVT 集

FirstLast 类用于 FIRSTVT 集合和 LASTVT 集合构造。

主要数据：

first：char[][] 用于存储非终结符的 FIRSTVT 集合；

last： char[][] 用于存储非终结符的 LASTVT 集合。

对 FIRSTVT 集的构造我们可以给出一个算法，这个算法基于下面两条规则：

（1） 若有产生式 A→a…或 A→Ba…，则 a 属于 FIRSTVT（A），其中 A，B 为非终结符，a 为终结符

（2） 若 a 属于 FIRSTVT(B)且有产生式 A→B…则有 a 属于 FIRSTVT(A)。

为了计算方便，我们建立一个布尔数组 F[m,n]（m 为非终结符个数，n 为终结符个数）和一个后进先出栈 STACK。我们将所有的非终结符排序，用 的序号，再将所有的终结符排序，用 表示终结符 a 的序号。算法的目的是要合数组每一个元素最终取什满足：F[,]的值为真，当且仅当 a 属于 FIRSTVT（A）。至此，显然所有非终结符的 FIRSTVT 集已完全确定。

步骤如下：

首先按规则（1）对每个数组元素赋初值。观察这些初值，若 F［i_A， j_a]的值为真，则将（A ,a）推入栈中，直至对所有数组元素的初值都按此处处理完。然后对栈做以下运算。

将栈顶项弹出，设为（B，a），再用规则（2）检查所有产生式，若有形为 A→B…的产生式，而 F［i_A，j_a]的值是假，则令其变为真，且将（A ,a）推进栈，如此重复直到栈弹空为止。具体的算法可用程序描述为：

```
PROCEDURE    INSERT（A，a）;
        IF  NOT  F［$i_A$，$j_a$］   THEN
            BEGIN
                F［$i_A$，$j_a$］:=TRUE
                PUSH(A,a)    ONTO    STACK
            END
```

此过程用于当 a 属于 FIRSTVT（A）时置 F［i_A，j_a］为真，并将符号对（A,a）下推到栈中，其主程序为：

```
BEGIN     （MAIN）
   FOR  I 从 1 到 m,j 从 1 到 n
       DO  F［$i_A$，$j_a$］：=FALSE;
    FOR  每个形如 A→a…或 A→Ba…的产生式
       DO  INSERT（A，a）
    WHILE    STACK 非空    DO
       BEGIN
           把 STACK 的顶项记为（B，a）弹出去
           FOR  每个形如 A→B…的产生式
              DO INSERT（A，a）
           END
END    （MAIN）
```

利用类似的方法可求得每个非终结符的 LASTVT（A）

由 LASTVT 和 FIRSTVT 集建立优先矩阵

Table 类利用之前构造的 LASTVT 和 FIRSTVT 生成。

主要数据：

table: int[][] 用于存储算符优先关系矩阵。

有了文法中的每个非终结符的 FIRSTVT 集和 LASTVT 集，我们就可以用如下算法最后构造文法的优先关系表

FOR 每个产生式 A→X_1 X_2…X_n DO

FOR i:=1 TO n-1 DO

BEGIN

IF X_i和X_{i+1}均为终结符

THEN 置X_i =X_{i+1}

IF X_i和X_{i+2}都为终结符，但X_{i+1}为非终结符

THEN 置X_i =X_{i+2}；

IF X_i为终结符而X_{i+1}为非中介符

THEN FOR FIRSTFVT(X_{i+1})中的每个 b DO 置 X_i <b；

IF X_i为非终结符而X_{i+1}为终结符

THEN FOR LASTVT(X_i)中的每个 a DO 置 a>X_{i+1}

END

以上算法对任何算符文法 G 可自动构造其算符优先关系表，并可判断 G 是否为算符优先关系

算符文法的分析归约过程算法

自底向上的算符优先分析法，也为自左右向右归约，我们已经知道它不是规范归约。规范归约的关键问题是如何寻找当前句型的句柄，句柄为某一产生式的右面部，归约结果为用与句柄相同的产生式右面部之左部非终结符代替句柄，而算符优先分析归约的关键，是如何找最左素短语，而最左右素短语$N_i\ \ a_i\ \ N_{i+1}\ a_{i+1}\cdots\cdots a_j\ \ N_{j+1}$应满足：

$a_{i-1}\ \langle\cdot\ \ a_i$

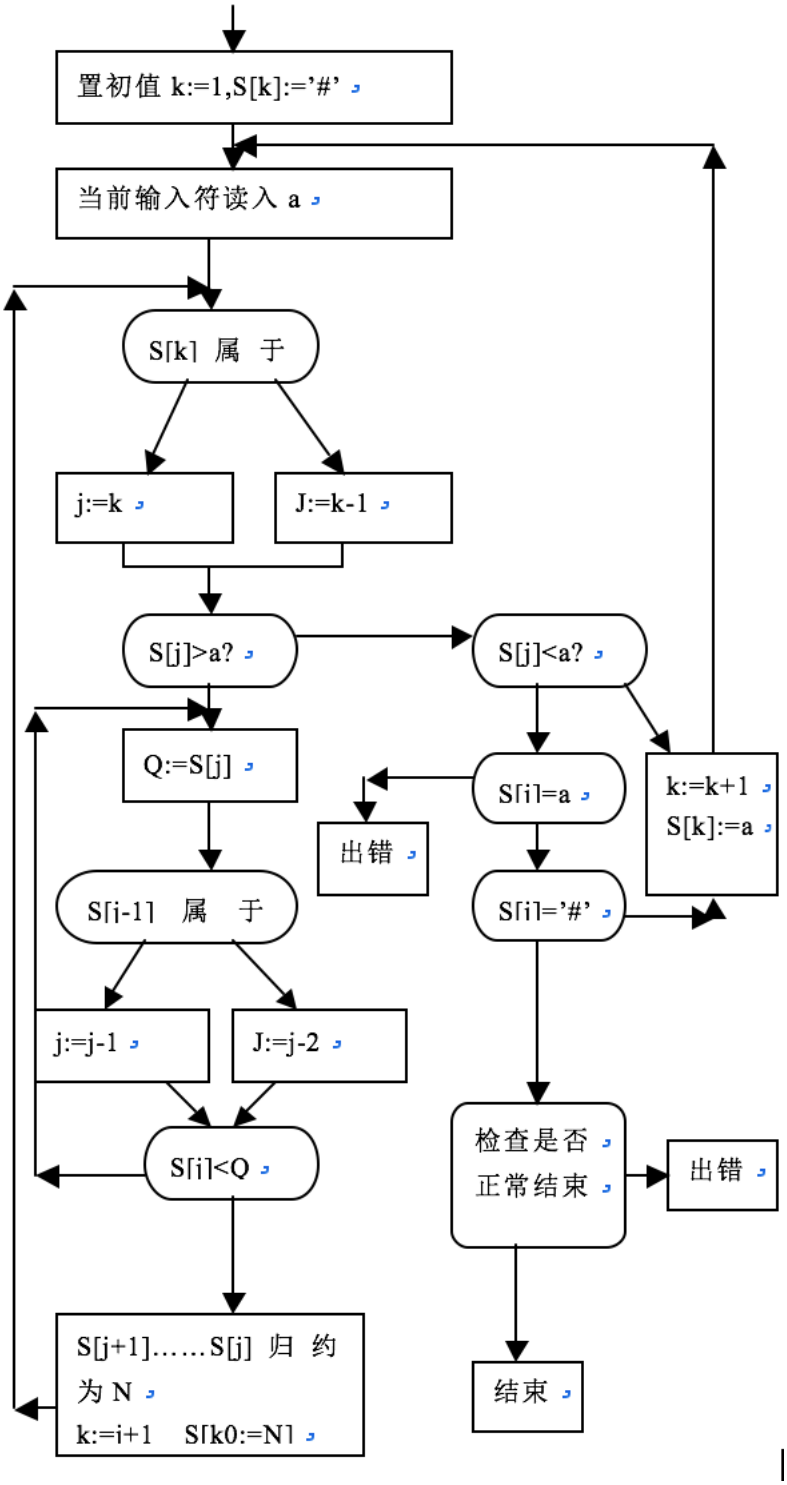
$a_i\ =\ a_{i+1}\ =\cdots\cdots a_j$

$a_j\ \ \cdot\rangle\ \ a_{j+1}$

在文法的产生式中存在右面部符号串的符号个数与该素短语的符号个数相等，非终结符号对应N_k，（k=i,…,j+1）不管其符号名是什么。终结符对应a_i,…，a_j,其符号表示要与实际的终结符相一致才有可能形成素短语。由此，我们在分析过程中可以设置一个符号栈 S，用以寄存归约或待形成最左素短语的符号串，用一个工作单元 a 存放当前读入的终结符号，归约成功的标志是当读到句子结束符#时，S 本中只剩#N，即只剩句子最左括号“#”和一非终结符 N。下面给出分析过程的示意图

在归约时要检查是否有对应产生式的右部与 S[j+1]…S[k]形式相符，（忽略非终结符名的不同）若有才可归约，否则出错。在这个分析过程中把“#”也放在终结符串中。

算符优先分析的移进规约流程图:


置初值 k:=1,S[k]:='#'
当前输入符读入 a
S[k] 属 于
j:=k
J:=k-1
S[j]>a?
S[j]<a?
Q:=S[j]
S[i]=a
出错
k:=k+1
S[k]:=a
S[j-1] 属 于
S[i]='#'
j:=j-1
J:=j-2
检查是否
正常结束
出错
S[i]<Q
S[j+1]……S[j] 归 约
为 N
k:=i+1 S[k0:=N]
结束

语法分析树

语法树是在对字串进行算符优先分析时同步生成的，一个子树对应一个最左速短语。语法树的每一次构建操作对应算符优先分析时的一个规约操作。

语法树根据算符优先规约的规则和逻辑，以自下而上的顺序生成

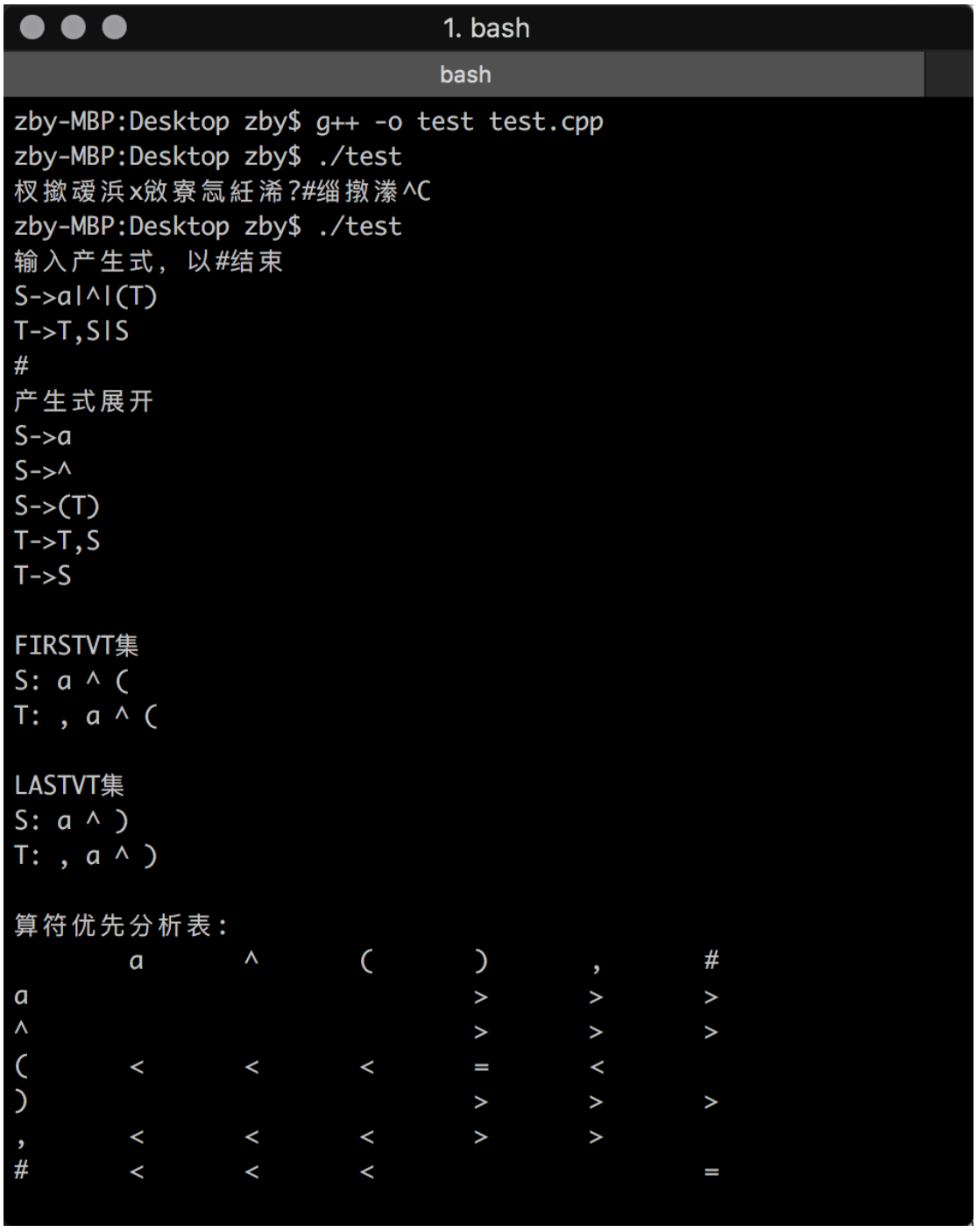
每次算符优先分析时要对 S[j+1]...S[k]规约为某个 N 时，记录下要规约的 S[j+1]...S[k]字符作为当前一个子树的叶子节点，N 作为当前子树的根节点建立父子关系。

将 S[j+1]...S[k]规约成的 N，存入一个栈中（代码中使用一个指针数组实现），保存下来，作为之后规约操作时一个子树的叶节点。

循环执行后，在最后一步规约时，将 S[j+1]...S[k]规约成的 N 作为语法树的根节点 root。

将建立好的语法树逻辑结构，带入多叉树建立算法中，生成语法树，并打印出来。

运行截图


1. bash
bash
zby-MBP:Desktop zby$ g++ -o test test.cpp
zby-MBP:Desktop zby$ ./test
权撳碍浜x敞寮氙紝浠?#缁撴潃^C
zby-MBP:Desktop zby$ ./test
输入产生式，以#结束
S->a|^|(T)
T->T,S|S
#
产生式展开
S->a
S->^
S->(T)
T->T,S
T->S
FIRSTVT集
S: a ^ (
T: , a ^ (
LASTVT集
S: a ^)
T: , a ^)
算符优先分析表：
a ^ ( ) , #
a > > >
^ > > >
(< < < = <
) > > >
, < < < > >
< < < =

```
输入单词串
(a,(a,a))#
栈                  输入串          动作
#                   (a,(a,a))#      移进
#(                  a,(a,a))#       移进
#(a                 ,(a,a))#        规约
#(S                 ,(a,a))#        移进
#(S,                (a,a))#         移进
#(S,(               a,a))#          移进
#(S,(a              ,a))#           规约
#(S,(S              ,a))#           移进
#(S,(S,             a))#            移进
#(S,(S,a            ))#             规约
#(S,(S,S            ))#             规约
#(S,(T              ))#             移进
#(S,(T)             )#              规约
#(S,S               )#              规约
#(T                 )#              移进
#(T)                #               规约
#S                  #               接受
accept
```

```
生成的语法树：
S
├─)
├─T
│  ├─S
│  │  ├─)
│  │  ├─T
│  │  │  ├─S
│  │  │  │  └─a
│  │  │  ├─,
│  │  │  └─S
│  │  │     └─a
│  │  └─(
│  ├─,
│  └─S
│     └─a
└─(
```

实验总结

在算符优先文法的程序设计过程中，程序比较复杂，光数据结构设计阶段就占据了很长的时间，几种数据结构很难确定那种能满足程序要求。一旦确定了数据结构，程序编写还是比较顺利。很快程序就可以对正确的输入做出正确的处理，不过没有查错纠错的能力，之后加入查错纠错的功能花费了很大的力气。对整个编程的过程把握的不是很好，以致调试很艰难。

本程序虽然还不能尽如人意，但本人也已经尽力把所有已知的错误作出修正，这次编程其实收获还是很大的。

首先，再一次锻炼了自己独立编程的能力。从整个要求分析，到数据据够设计和一些算法的实现都使我对程序设计有了新的设计体会。

其次，通过该课程设计，全面系统的理解了编译原理程序构造的一般原理和基本实现方法。把学过的计算机编译原理的知识强化，能够把课堂上学的知识通过自己设计的程序表示出来，加深了对理论知识的理解，通过实际动手做实验，从实践上认识了操作系统是如何处理命令的，对计算机编译原理的认识更加深刻。