编译原理实验报告

算符优先文法分析

姓　　名：　　牟鑫一

学　　号：　　20161001764

班　　级：　　191174

指 导 老 师：　　刘远兴

目　　录

一、 题目

算符优先文法分析

二、 问题描述

- 根据给定文法，先求出 FIRSTVT 和 LASTVT 集合，构造算符优先关系表（要求算符优先关系表输出到屏幕或者输出到文件）；
- 根据算法和优先关系表分析给定表达式是否是该文法识别的正确的算术表达式（要求输出归约过程）；
- 给定表达式文法为：

```
GE '): E'→#E#
E->E+T|T
T->T*F|F
F->E)|i
```

- 分析的句子为：

  i+i)*i 和 i+i)*i

三、 基本要求

- 选择最有代表性的语法分析方法算符优先法；
- 选择对各种常见程序语言都用的语法结构，如赋值语句（尤指表达式）作为分析对象，并且与所选语法分析方法要比较贴切；
- 实习时间为 6 学时。

四、 小组分工

- 牟鑫一：根据给定文法，求出 FIRSTVT 集；
- 江佳盛：根据给定文法，求出 LASTVT 集；

- 刘栋阳：构造算符优先关系表；
- 郭兴：根据算法和优先关系表分析给定表达式是否该文法识别的正确的算术表达式，输出归约过程；
- 李泽栋、梅涵：综合，编写整个程序的框架、界面设计、数据结构设计等。

五、 整体设计

1、 正则文法

- 用户需输入正则文法的规则个数，以便于对正则规则的遍历；
- 由用户输入正则文法，可识别“或”关系（即“|”），输入的正则规则存储到二维数组 str[maxn][Maxn]中；
- 用一个双层循环获取输入的正则规则里包含的终结符，存储到数组 terminator[maxn]中备用。

2、 获取 FIRSTVT 集

- 遍历正则规则的左部，依次查找每个非终结符的 FIRSTVT 集；
- 对应某一个正则规则的左部非终结符，遍历这条正则规则的每个字符，对正则规则的右部做判断，找出该正则规则对应左部非终结符的 FIRSTVT 集。

3、 输出 FIRSTVT 集

- 遍历数组 firstvt[maxn][maxn]输出各个非终结符的 FIRSTVT 集

4、 数据结构

由于是但文件结构，所以可直接使用全局变量：

```
const int Maxn = 110;        //单条规则最多 110 个字符
const int maxn = 20;         //最多 20 条文法规则
```

```
char str[maxn][Maxn];        //储存输入的正则文法
char terminator[maxn];       //存储终结符
char firstvt[maxn][maxn];    //存储 FIRSTVT 集
int firstflag[maxn];         //记录非终结符的 FIRSTVT 集是否已求出
int fcnt[maxn];              //非终结符 FIRSTVT 集的元素个数
```

5、 文件结构

- Operator-precedence Parsing.cpp

6、 基本思想

对应某一个正则规则的左部非终结符，遍历这条正则规则的每个字符，判断右部的第一个字符是否终结符，若是则将其加入该非终结符的 FIRSTVT 集；若不是则判断第二个元素是否终结符，若是则将其加入该非终结符的 FIRSTVT 集；若右部第一个字符是自身，则继续往后判断；若右部第一个字符是不是自身的其它非终结符，则递归获取这个非终结符的 FIRSTVT 集并加入该非终结符的 FIRSTVT 集；继续往后遍历这一正则规则的后续字符，查找是否有符号“|”，若有则作上述相同的判断。

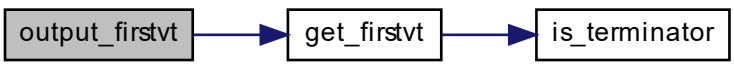
如上直至遍历完该字符串，表明该非终结符的 FIRSTVT 集已找全，已找全的非终结符在数组 firstflag[maxn]中标记为 1，当再次遇到该非终结符时则不用重复求 FIRSTVT 集。

六、 自己负责的模块设计

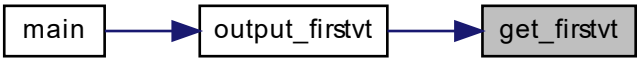
我负责的模块是：根据给定文法，求出 FIRSTVT 集

1、 函数调用关系图

output_firstvt） 函数调用 get_firstvt） 函数获取某一正则规则的左部非终结符的 FIRSTVT 集，get_firstvt） 函数调用 is_terminator） 函数判断某字符是否终结符：



Main）函数调用 output_firstvt） 函数输出所有非终结符的 FIRSTVT 集：



2、 模块接口说明

1、 void output_firstvt(int T);

传入正则规则的条数 T，输出 FIRSTVT 集

2、 void get_firstvt(char s, int T);

传入非终结符 s，正则规则条数 T，获取非终结符 s 的 FIRSTVT 集

3、 int is_terminator(char c);

传入字符 c，判断其是否为终结符，是返回 1，不是返回 0

3、 函数的功能实现

```
//判断字符 c 是否终结符
int is_terminator(char c);
//输出 firstvt 集
void output_firstvt(int T);
//求 s 非终结符的 FIRSTVT 集
//参数 s 为正则文法的左部非终结符，T 为正则规则的个数
void get_firstvt(char s, int T);
```

七、 算法设计

1、 判断字符 c 是否终结符

```
int is_terminator(char c) { //判断字符 c 是否终结符
    for (int i = 0; terminator[i] != '\0'; i++) {
        if (terminator[i] == c)
            return 1;
    }
    return 0;
}
```

2、 求 s 非终结符的 FIRSTVT 集

```
//参数 s 为正则文法的左部非终结符，T 为正则规则的个数
void get_firstvt(char s, int T) {
    int i, j, tt;
    //该 for 循环找出非终结符 s 所在正在规则的序号 i
    for (i = 0; i < T; i++) {
        if (str[i][0] == s)
            break;
    }
    //如果 str[i][0]的 FIRSTVT 集未求出，
    if (!firstflag[i]) {
        int k = fcnt[i]; //str[i][0]的 FIRSTVT 集元素个数，初值为 0
        //遍历正则规则 str[i]的每个字符
        for (j = 0; str[i][j] != '\0'; j++) {
            if (str[i][j] == '>' | str[i][j] == '|') {
                //判断后一个字符是否终结符，若是则加入终结符集
                if (is_terminator(str[i][j + 1])) {
                    firstvt[i][k++] = str[i][j + 1];
                }
                else {
                    //否则判断其后第二个字符是否终结符，若是则加入终结符集
                    if (is_terminator(str[i][j + 2])) {
                        firstvt[i][k++] = str[i][j + 2];
                    }
                    //如果后一个非终结符不是自身，则需要递归获取该非终结符的 FIRSTVT 集加
入到 str[i][0]的非终结符集中
                    if (str[i][j + 1] != s) {
                        int ii, jj;
                        //递归获取非终结符 str[i][j + 1]的 FIRSTVT 集
                        get_firstvt(str[i][j + 1], T);
                        //该 for 循环找出非终结符 str[i][j+1]所在正在规则的序号 ii
                        for (ii = 0; ii < T; ii++) {
                            if (str[ii][0] == str[i][j + 1])
                                break;
                        }
                       //将非终结符 str[i][j + 1]的非终结符集中的元素加入到 str[i][0]
的 FIRSTVT 集中
                        for (jj = 0; jj < fcnt[ii]; jj++) {
                            for (tt = 0; tt < k; tt++) {
                                if (firstvt[i][tt] == firstvt[ii][jj])
                                    break;
                            }
                            //tt == k 表明终结符 firstvt[ii][jj]应加入 FIRSTVT 集
```

```
                            if (tt == k) {
                                firstvt[i][k++] = firstvt[ii][jj];
                            }
                        }
                    }
                }
            }
        }
        firstvt[i][k] = '\0';//str[i][0]的 FIRSTVT 集已查找完毕，添加空串作为串结束符
        fcnt[i] = k;//记录 str[i][0]的 FIRSTVT 集元素个数
        firstflag[i] = 1;//标记已获取到 str[i][0]的 FIRSTVT 集
    }
}
```

3、 输出 FIRSTVT 的函数

```
void output_firstvt(int T) { //输出 firstvt 集
    for (int i = 0; i < T; i++) {
        get_firstvt(str[i][0], T);//获取 str[i][0]的 FIRSTVT 集
    }
    for (int i = 0; i < T; i++) {
        printf("FIRSTVT[%c]:", str[i][0]);
        for (int j = 0; j < fcnt[i]; j++) {//fcnt[i]为 FIRSTVT 集元素个数
            printf("%c ", firstvt[i][j]);
        }
        puts("");//将空字符串写入到标准输出 stdout，并追加一个换行符，等效于输出一个换行
    }
}
```

八、 调试分析

1、 优点分析

获取 FIRSTVT 集的函数考虑比较全面，综合考虑了各种情况，使用多个 if 语句进行判断，可以处理带有特殊符号或“|”的正则规则，采用递归处理了开头为非终结符的问题，一个函数就可以应付多层推导的问题

2、 缺点分析

- 数据存储采用的是全局普通变量，对于我们在做的程序是没有什么问题的，但是全局变量局限太多，不适用于大型项目，不利于程序的扩展；
- 获取 FIRSTVT 集的函数 get_firstvt) 有些复杂，嵌套了多层选择结构和循环结构，应该是可以再精简些或者将一些代码独立出来写成单独的函数来调用。

3、 改进方法

- 可以将一些数据和函数抽象到类中，增强可读性和扩展性；
- 较为复杂的函数可以模块化处理，拆分为多个小函数，通过函数调用实现同样的功能，是程序更容易理解。

九、 使用手册

从全局变量中获取正则文法数据，外部只需调用 output_firstvt) 即可输出所有非终结符的 FIRSTVT 集，output_firstvt) 函数调用 get_firstvt) 函数获取某一非终结符的 FIRSTVT 集，将各非终结符的 FIRSTVT 集中的元素的个数存储到 fcnt[]数组中，用 firstflag[maxn]数组标记某一非终结符的 FIRSTVT 集是否已求出，将求出的 FIRSTVT 集存储到数组 firstvt[maxn][maxn] 中。

遍历二维数组 firstvt[maxn][maxn] 即可获取各非终结符的 FIRSTVT 集元素。

十、 测试结果

输入正则规则个数：3

输入正则文法：

```
E->E+T|T
T->T*F|F
F->(E)|i
```

执行结果如下：

```
输入正则文法个数：3

输入第1条正则文法：E->E+T|T
输入第2条正则文法：T->T*F|F
输入第3条正则文法：F->(E)|i

输出该文法的FIRSTVT集如下：

FIRSTVT[E]:+ * ( i
FIRSTVT[T]:* ( i
FIRSTVT[F]:( i
```

十一、 总结

首先通过这个题目，我对算符优先分析有了更加深刻的理解，对 FIRSTVT 集的定义理解得更加透彻，搞清楚了求解 FIRSTVT 集以及 LASTVT 集得详细步骤。

其次，通过这两个题目的联系，我对编译原理学过的内容都有了更多自己的理解，对我们学习编程以来一直在用的各种编译器也有了一些自己的理解，收获颇多。