编译原理大作业——算符优先分析表

1. 使用说明

- 实现环境为`windows`,所用编程语言为`C++`,应要求未使用`STL`。
- 将压缩包解压，得到三个文件：`OFG.cpp`，`test.txt`和`OFG.exe`。其中`OFG.cpp`是源代码，`OFG.exe`是`OFG.cpp`经过`g++`编译得到的可执行文件，`test.txt`是`OFG.exe`程序的输入文件。
- 若要重新利用源代码生成可执行文件可以在`windows`的`cmd`下输入`g++ -o OFG ./OFG.cpp`
- 为了程序能正常运行请不要修改`test.txt`文件的文件名，并确保`test.txt`文件和程序在同一层级的目录下。若要测试不同文法，请将文法产生式复制粘贴到`test.txt`文件内，覆盖原有文法。

2. 对输入文法的要求

本程序有一定局限性，对输入文法有一些要求，下面用例子来说明：

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

上面文法的输入格式是本程序能够接受的标准格式，其余格式不被支持。上面的输入格式包括：

1. 同一非终结符号对应的产生式需要写成一行，之间用`|`隔开，例如，下面的格式不被支持：

   ```
   E -> E + T
   E -> T
   ```

2. 任何两个有完整意义的字符之间必须要用空格隔开，包括用来分隔不同产生式的`|`符号，都需要和其前后两个有完整意义的字符用空格隔开，例如，下面的格式不被支持：

   ```
   E->E+T|T //相邻字符之间没有空格
   ```

   ```
   T -> T*F |F // |符号与其前后字符间没有空格
   ```

   ```
   F -> (E) | i //(, E, )是三个有完整意义的字符，需要用空格隔开
   ```

3. 本程序可以判断输入文法是否属于算符优先文法，只能对算符优先文法进行分析表的构建。由于ϵ不易表示，在程序实现过程中使用`#`表示空串。

3. 系统实现简述

算符优先分析表的构建算法虽然理解起来并不困难，但若想利用一门编程语言实现，就应该处理好以下几个方面：

1. 对输入文法的处理，实质是对字符串的处理
2. 如何表示产生式，实质是对数据结构的选择
3. 如何计算非终结符号的FIRSTVT，LASTVT集合，实质是对递归函数的实现
4. 如何构建算符优先分析表
5. 对结果的输出，实质是字符串格式控制

本程序采用的编程语言是C++，由于老师规定了若使用C++，则不能借助STL里面现有的工具，因此在字符串处理相关工作方面会显得繁琐。源文件中用于对输入文法进行处理的函数有：

```
int getInput(char** buffer);
// getInput函数用来获取输入文件一共有多少行
void getProductions(char** buffer, production* prod, const int& cnt);
// getProductions函数用来获取每一行的产生式
void getVN(char** VN, production* prod, const int& cnt);
// getVN函数利用获取到的产生式确定哪些符号是非终结符
int getVT(char** VT, char** VN, production* prod, const int& cnt, char***
table);
// getVT函数利用获取到的产生式和非终结符号确定哪些符号是终结符
bool isVT(char* s, char** VT, int cnt1);
// 给定一个符号，判断其是否属于终结符号
bool check(production* prod, const int& cnt, char** VT, int cnt1);
// 在获取了产生式，非终结符集和终结符集之后，用于判断输入文法是否是算符优先文法
```

本程序对产生式的表示选择用一个`production`类来进行，其具体定义如下：

```
class production
{
public:
    char* left; //每一行的左部，其实就是非终结符号
    char** right; //由于每一行的右部其实是多个用 | 隔开的产生式，所以需要二维字符串数组
    int rows; //记录每一行右部有多少产生式
    production() //构造函数
    {
        left = new char[MAX];
        right = new char* [MAX];
        for (int i = 0; i < MAX; ++i)
        {
            right[i] = new char[MAX];
        }
        rows = 0;
    }
    production(char* l, char** r, int row) //构造函数
    {
        rows = row;
        strcpy(left, l);
        right = new char* [row];
        for (int i = 0; i < row; ++i)
        {
            strcpy(right[i], r[i]);
        }
    }
    void print() //打印相应一行产生式
    {
        printf("%s -> %s", left, right[0]);
        for (int i = 1; i < rows; ++i)
        {
            printf(" | %s", right[i]);
        }
        cout << endl;
    }
};
```

对非终结符号的FIRSTVT和LASTVT的计算，程序中采用的是递归的方法，由于相关函数较长，这里只取其中一个通过注释进行简单说明：

```
int DFS(int x, production* prod, char** VT, int cnt, int cnt1, char*** FIRSTVT,
char*** SET)
{
    // 这里x代表第x个非终结符号
    //若已经计算过，返回即可
    if (vis[x])
        return numF[x];
    vis[x] = 1; //标记已经计算
    int num = 0;
    int idx = 0;
    for (int i = 0; i < prod[x].rows; ++i)//对第x个非终结符的每个产生式右部
    {
        int nt = 0;
        char* number1 = new char[MAX];
        char* number2 = new char[MAX];
        char** tmp = new char* [MAX];
        number1[0] = char(i + '0');
        number1[1] = '\0';
        number2[0] = char(i + 1 + '0');
        number2[1] = '\0';
        for (int j = 0; j < MAX; ++j)
        {
            tmp[j] = new char[MAX];
        }
        for (int j = idx; j < NT[x]; ++j)
        {
            if (strcmp(SET[x][j], number1) == 0)
                continue;
            if (strcmp(SET[x][j], number2) == 0)
            {
                idx = j;
                break;
            }
            strcpy(tmp[nt], SET[x][j]);
            nt++;
        }
        if (nt == 0)
        {
            cout << "error!";
            exit(-1);
        }
        if (isVT(tmp[0], VT, cnt1))//对应于算法中的P -> a....
        {
            strcpy(FIRSTVT[x][num], tmp[0]);
            num++;
        }
        else //第一个符号不是终结符
        {
            if (nt > 1 && isVT(tmp[1], VT, cnt1)) //对应于算法中的：P -> Qa...
            {
                strcpy(FIRSTVT[x][num], tmp[1]);
                num++;
            }
            int y = 0;
            for (int k = 0; k < cnt; ++k)
            {
                if (strcmp(tmp[0], prod[k].left) == 0)
                {
```

```
                    y = k;
                    break;
                }
            }
            if (y != x)
            {
                int numy = DFS(y, prod, VT, cnt, cnt1, FIRSTVT, SET);
                for (int k = 0; k < numy; ++k)
                {
                    bool flag = true;
                    for (int kk = 0; kk < num; ++kk)
                    {
                        if (strcmp(FIRSTVT[x][kk], FIRSTVT[y][k]) == 0)
                        {
                            flag = false;
                            break;
                        }
                    }
                    if (flag)
                    {
                        strcpy(FIRSTVT[x][num], FIRSTVT[y][k]);
                        num++;
                    }
                }
            }
        }
    }
    numF[x] = num;
    return num;
}
```

得到了每个非终结符号的FIRSTVT和LASTVT之后，构建算符优先分析表只需要按照算法按部就班执行即可，值得注意的是`$`符号和所有终结符号的优先关系，需要添加增广文法来分析。最后，对结果的输出也只是对字符串格式的控制，较为繁琐，却并不困难，不再赘述。

4. 总结与感悟

本次大作业使得我对算符优先分析表的构建算法有了更好地理解与掌握，但是由于无法使用STL里面现有的工具，在实现过程中确实要走许多弯路，而且实现不够简洁，高效。