编号：_____

实习	一	二	三	四	五	六	七	八	九	十	总评	教师签名
成绩												

武汉大学网络安全学院

《编译原理》课程

文法分类器

实习报告

编　　号：CP2023 NS2WE

实习题目：文法分类器

专业（班）：信息安全（5 班）

学生学号：2021302181126

学生姓名：谢锋

任课教师：杜　卓　敏

2023 年　4 月　3 日

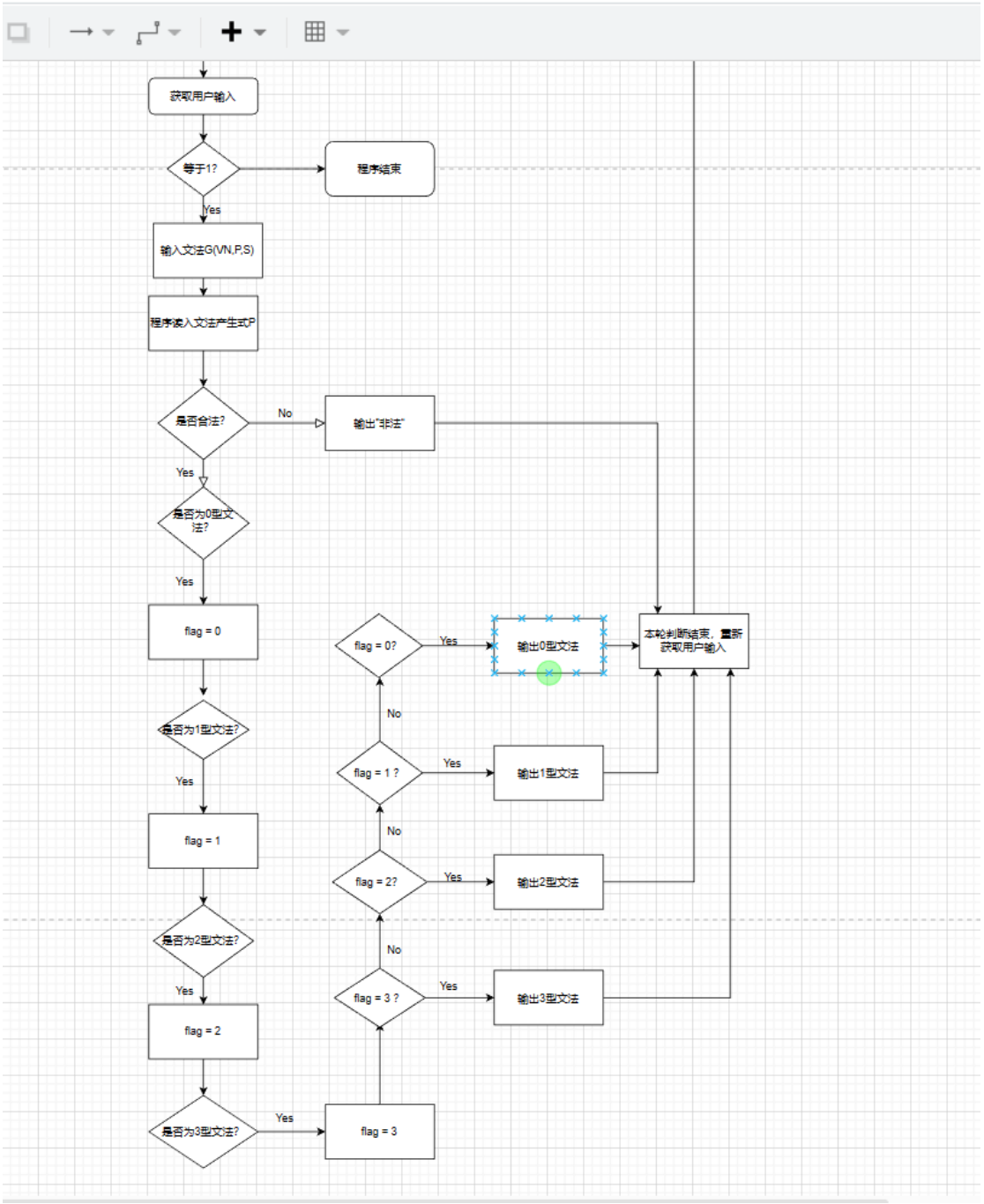
目　　录

第一部分　文法分类算法（过程描述即可，无需代码）
…… 1
第二部分　测试计划 …… 11
第三部分　测试报告 …… 12

第一部分 文法分类算法

一、检测流程

流程图如下：



1、 为了实现用户友好，我们让用户决定是否查询：输入 1 即可查询文法的类型，输入 0 即可退出程序。

2、 当输入为 1 时，程序会提示用户输入文法 G，然后程序会对输入的文法进行类型判断，并输出相应的文法

3、 一次输出完成后，程序会重复步骤一，直到用户输入 0

二、 代码详解

1、 我们使用 string 类存储非终结符、终结符、以及文法产生式，所以需要对输入的字符串进行处理。由于我们使用逗号分隔每个非终结符或文法产生式，故我们需要使用 split 函数对字符串进行分割。C++没有这个函数，需要我们手动实现：

```cpp
// 分割输入的字符串：以逗号作为分隔符
vector<string> split(const string &str, const string &delim)
{
    vector<string> res;
    if ("" == str)
        return res;
    char *strs = new char[str.length() + 1];
    strcpy(strs, str.c_str());
    char *d = new char[delim.length() + 1];
    strcpy(d, delim.c_str());
    char *p = strtok(strs, d);
    while (p)
    {
        string s = p;
        res.push_back(s);
        p = strtok(NULL, d);
    }
    return res;
}
```

其中，delim 的定义如下：

```cpp
// 分隔符
const char *delim = ",";
```

2、判断输入的产生式是否合法。我们这样定义一个文法是否合法：若某条文法产生式中出现了非终结符中没有的符号，我们就说这个文法是非法的。所以我们应当首先判断输入的文法是否合法，若不合法，直接结束：

```cpp
bool check(const Grammar &G)
{
    vector<string> tmp = G.VN;
    unordered_map<string, int> m;
    for (auto x : tmp)
        m[x]++;
    for (auto x : G.P)
    {
        for (int i = 0; i < x.length(); i++)
        {
            string t = "";
            t = x[i];
            if (x[i] >= 'A' && x[i] <= 'Z' && !m[t])
                return false;
        }
    }
    return true;
}
```

3、判断文法的类型

1）0 型文法

这类文法的判断比较简单，我们只需要判断产生式的左部是否有非终结即可：如果没有非终结符，直接返回 false，否则返回 true。

```cpp
bool JudgeZero(const Grammar &G)
{
    bool flag = false;
    vector<string> tmp = G.P;
    for (string str : tmp)
    {
        // 扫描每一条文法产生式，并判断其左部是否存在非终结符
        for (int i = 0; i < str.length(); i++)
        {
            if (str[i] == ':' && str[i + 1] == ':' && str[i + 2] == '=')
            {
                for (int j = 0; j < i; j++)
                {
                    if (str[j] >= 'A' && str[j] <= 'Z')
                    {
                        flag = true;
                        break;
                    }
                }
                break;
            }
        }
    }
    // 若没有非终结符，返回假
    return flag;
}
```

2） 1 型文法

1 型文法的定义如下：产生式左部的长度小于等于产生式右部的长度，或者特殊情况：S-> ε ，据此，我们编写的函数如下：

```cpp
bool JudgeOne(const Grammar &G)
{
    // left记录产生式的左部
    // right记录产生式的右部
    bool flag = false;
    string left;
    string right;
    for (string str : G.VN)
    {
        for (int i = 0; i < str.length(); i++)
        {
            if (str[i] == ':' && str[i + 1] == ':' && str[i + 2] == '=')
            {
                left = str.substr(0, i);
                right = str.substr(i + 3, str.length() - i - 3);
            }
            if (right.length() >= left.length())
                flag = false;
            else if (left.length() == 1 && right == "ε")
                flag = true;
            else
                return false;
        }
    }
    return flag;
}
```

3） 2 型文法

这类文法的判断最简单，若某文法满足 1 型文法的条件，我们只需要再判断产生式的左部是否只有一个非终结符即可：

```
bool JudgeTwo(const Grammar &G)
{

    string left = "";
    for (auto str : G.P)
    {
        for (int i = 0; i < str.length(); i++)
        {
            if (str[i] == ':' && str[i + 1] == ':' && str[i + 2] == '=')
            {
                left = str.substr(0, i);
                if (left.length() > 1)
                    return false;
            }
            break;
        }
    }
    return true;
}
```

4） 3 型文法

3 型文法的判断条件如下：

(1) 产生式的左部只有一个非终结符

(2) 产生式的右部只有一个终结符或者非终结符

(3) 当产生式的右部有两个符号时，必须是一个为终结符，一个为非终结符，并且每条产生式的线性要完全相同

可见 3 型文法的判断还是比较复杂，我的代码如下（代码太长，不传图片）：

```
1.bool JudgeThree(const Grammar &G)
2.{
3.    string right = "";
4.    // flag 数组记录所有产生式的线性：1 表示左线性，2 表示右线性
5.    int idx = 0;
6.    int flag[N] = {0};
7.    for (auto str : G.P)
8.    {
9.        for (int i = 0; i <= str.length(); i++)
10.        {
11.            if (str[i] == ':' && str[i + 1] == ':' && str[i + 2] == '
    =')
12.            {
13.                right = str.substr(i + 3, str.length() - i - 3);
14.                if (right.length() == 0)
15.                    return false;
16.                if (right.length() == 2)
17.                {
18.                    // 产生式右部是两个非终结符，返回假
19.                    if (right[0] >= 'A' && right[0] <= 'Z' && right[1
    ] >= 'A' && right[1] <= 'Z')
20.                        return false;
21.                }
22.                if (right.length() > 2 && right.length() <= 5)
```

```
23.                 {
24.                     int t = right.find('|');
25.                     if (t == -1)
26.                         return false;
27.                     string l = right.substr(0, t);
28.                     string r = right.substr(t + 1, right.length() - t - 1);
29.                     // val_l 和 val_r 记录左线性还是右线性：1 表示左线性，2 表示右线性
30.                     // flag1 用来记录特殊情况：l 或 r 有一个长度为 1，则 flag1=1
31.                     // cout << l << r << endl;
32.                     int val_l = 0, val_r = 0;
33.                     int flag1 = 0;
34.                     if (l.length() == 2)
35.                     {
36.                         if (l[0] >= 'A' && l[0] <= 'Z' && l[1] >= 'A' && l[1] <= 'Z')
37.                             return false;
38.                         if (l[0] >= 'A' && l[0] <= 'Z')
39.                             val_l = 1;
40.                         else
41.                             val_l = 2;
42.                     }
43.                     if (r.length() == 2)
44.                     {
45.                         if (r[0] >= 'A' && r[0] <= 'Z' && r[1] >= 'A' && r[1] <= 'Z')
46.                             return false;
47.                         if (r[0] >= 'A' && r[0] <= 'Z')
48.                             val_r = 1;
49.                         else
50.                             val_r = 2;
51.                     }
52.                     if (l.length() == 1 || r.length() == 1)
53.                         flag1 = 1;
54.                     // 左右线性不一致直接返回假
55.                     if (val_l != val_r && !flag1)
56.                         return false;
57.                     // 用数组记录下当前文法产生式的线性
58.                     // S->Ab|B
59.                     // 此时 val_l 为 1，但是 val_r 为 0，所以是左线性
60.                     flag[idx++] = val_l == 0 ? val_r : val_l;
61.                 }
```

```
62.                 if (right.length() > 5)
63.                 {
64.                     // 若当前产生式右部有多个|链接，我们需要依次判断每两个|之间的式子是否符合要求
65.                     // 例如：S::=Sa|b|c|d
66.                     vector<string> t = split(right, "|");
67.                     // 与 flag 数组一样，v 数组记录这些子表达式是左线性还是右线性的
68.                     int cnt = 0;
69.                     int v[N] = {0};
70.                     for (int j = 0; j < t.size(); j++)
71.                     {
72.                         // cout << t[j];
73.                         if (t[j].size() > 2)
74.                             return false;
75.                         string tmp = t[j];
76.                         if (tmp.length() == 2)
77.                         {
78.                             if (tmp[0] >= 'A' && tmp[0] <= 'Z' && tmp[1] >= 'A' && tmp[1] <= 'Z')
79.                                 return false;
80.                             if (tmp[0] >= 'A' && tmp[0] <= 'Z')
81.                                 v[cnt] = 1;
82.                             else
83.                                 v[cnt] = 2;
84.                         }
85.                         cnt++;
86.                     }
87.                     for (int i = 0; i < cnt - 1; ++i)
88.                     {
89.                         if (v[cnt] != v[cnt + 1] && v[cnt] != 0 && v[cnt + 1] != 0)
90.                             return false;
91.                     }
92.                     flag[idx] = v[cnt - 1];
93.                 }
94.                 break;
95.             }
96.         }
97.     }
98.     for (int i = 0; i < idx - 1; ++i)
99.     {
100.         if (flag[i] != flag[i + 1] && flag[i] != 0 && flag[i + 1] != 0)
```

```
                return false;
        }
        return true;
    }
```

5） 每种类型的文法判断函数我们已经完成，最后只需要调用即可：

```
int cmp(const Grammar &G)
{
    // 非法的文法返回-1
    int flag = -1;
    if (!check(G))
        return flag;
    if (JudgeZero(G))
        flag = 0;
    if (JudgeOne(G) && flag == 0)
        flag = 1;
    if (JudgeTwo(G) && flag == 1)
        flag = 2;
    if (JudgeThree(G) && flag == 2)
        flag = 3;
    return flag;
}
```

4、 完成主函数。

在主函数中，我们使用循环实现用户与程序的交互，由用户决定是否要判断产生式的类型；同时，我们使用 c++中的流与文件操作，实现对输入和输出的处理（这更符合实际情况）；另外，主函数还肩负着接受用户输入已及实现程序输出的功能。

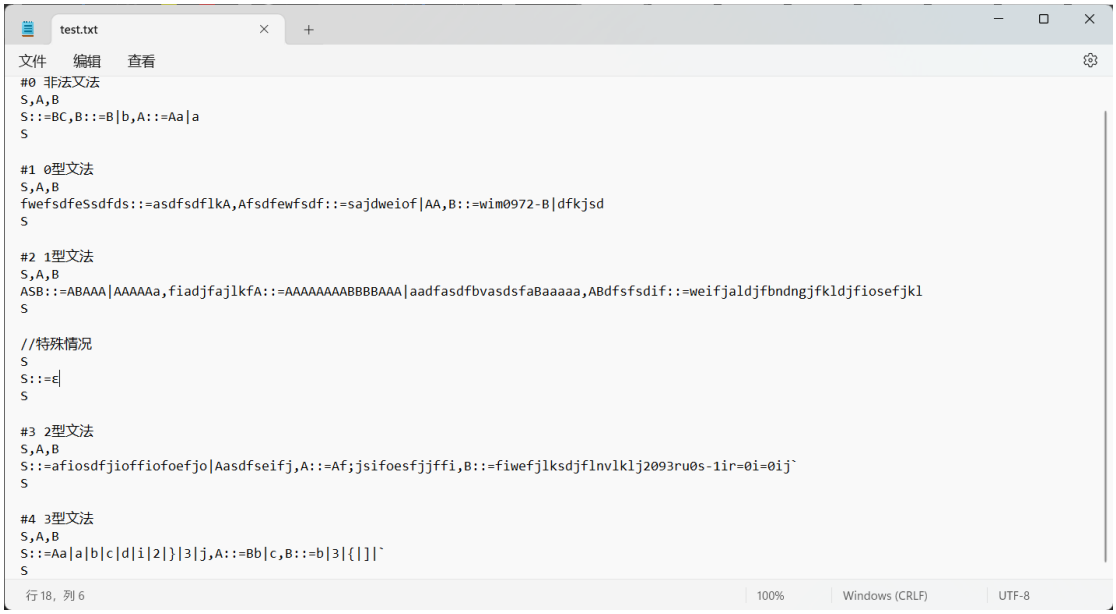
```
int main()
{
    int flag = 1;
    while (flag)
    {
        system("cls");
        cout << "输入 1 进行文法检测，输入 0 退出： " << endl;
        cin >> flag;
        if (flag == 0)
            break;
        system("cls");
        cout << "请按照提示输入文法 G： " << endl;
        cout << endl;
        cout << "文法输入格式：\n1、第一行输入非终结符\n2、第二行输入文法产生式\n3、第三行输入开始符号" << endl;
        cout << endl;
        cout << "例如：\n\tS,A,B\n\tS::=a|A,A|Bb,B::=b\n\tS\n";
        ofstream ofs;
        ofs.open("text.txt", ios::out | ios::trunc);
        for (int i = 0; i < 3; i++)
        {
            if (i == 0)
                cout << "请输入非终结符(以逗号隔开;英文大写字母)： ";
            else if (i == 1)
```

```
24.                 cout << "请输入文法产生式 P(以逗号隔开每个产生式)：";
25.             else if (i == 2)
26.                 cout << "请输入开始符号：";
27.             string tmp;
28.             cin >> tmp;
29.             ofs << tmp << endl;
30.         }
31.         ofs.close();
32.
33.         Grammar G;
34.         ifstream ifs;
35.         ifs.open("text.txt", ios::in);
36.         if (!ifs.is_open())
37.         {
38.             cout << "文件打开失败！" << endl;
39.             system("pause");
40.             return 0;
41.         }
42.         // buf1——buf4 分别存储：非终结符，终结符，文法产生式，开始符号
43.         string buf1, buf2, buf3, buf4;
44.         getline(ifs, buf1);
45.         getline(ifs, buf3);
46.         getline(ifs, buf4);
47.         G.VN = split(buf1, delim);
48.         G.P = split(buf3, delim);
49.         G.S = buf4;
50.         // buf 存储所有的终结符，且以逗号隔开
51.         string buf = "";
52.         unordered_map<char, int> mp;
53.         for (auto str : G.P)
54.         {
55.             for (int i = 0; i < str.length(); ++i)
56.             {
57.                 if ((str[i] < 'A' || str[i] > 'Z') && str[i] != ':' &
    & str[i] != '=' && str[i] != '|' && !mp.count(str[i]))
58.                 {
59.                     mp[str[i]]++;
60.                     buf += str[i];
61.                     buf += ",";
62.                 }
63.             }
64.         }
65.         // 去掉 buf 串的最后一个逗号，将前面的长度为 n-1 的字串赋值给 buf2
66.         buf2 = buf.substr(0, buf.length() - 1);
```

```
        G.VT = split(buf2, delim);
        int flag = cmp(G);
        cout << endl;
        printf("您输入的文法 G 是:\n");
        // printf只能输出c语言支持的内容,所以要把buf1的首地址指针取出来
        // 否则会输出乱码
        printf("G[%s] = ({%s}, {%s}, P, %s)\n", G.S.c_str(), buf1.c_str(), buf2.c_str(), G.S.c_str());
        printf("其中,   P为\n");
        for (auto x : G.P)
        {
            printf("\t");
            cout << x << endl;
        }
        if (flag == -1)
            cout << "非法! " << endl;
        else
            printf("该文法是 Chomsky%d 型文法\n", flag);
        cout << endl;
        ifs.close();
        system("pause");
    }
    cout << "欢迎下次使用! " << endl;
    system("pause");
    return 0;
}
```

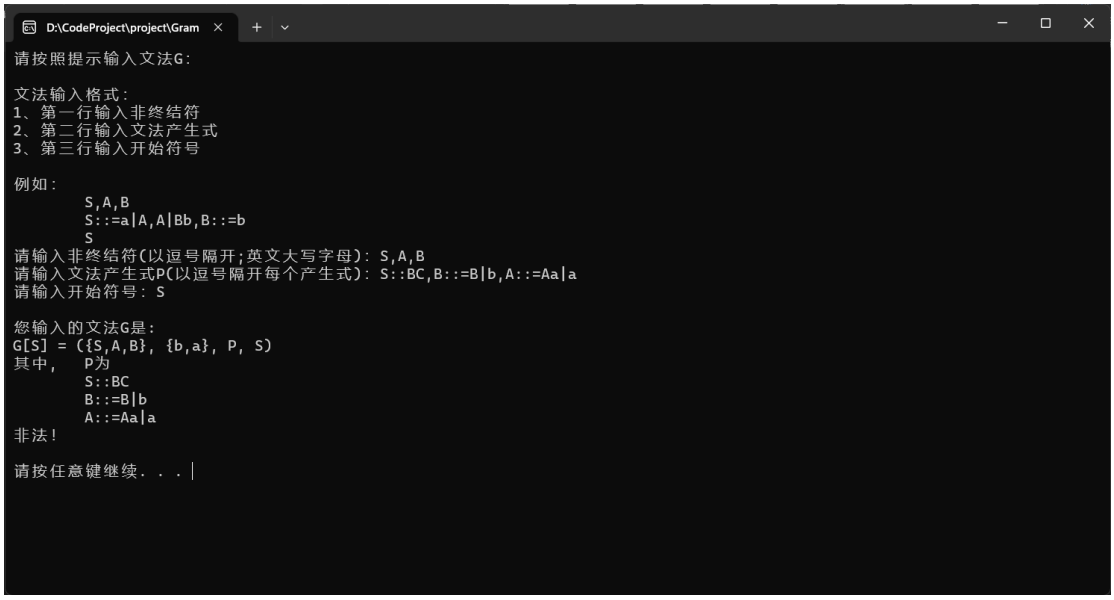
第二部分 测试计划

测试样例如下（该文件我会一并上交以供检查）：

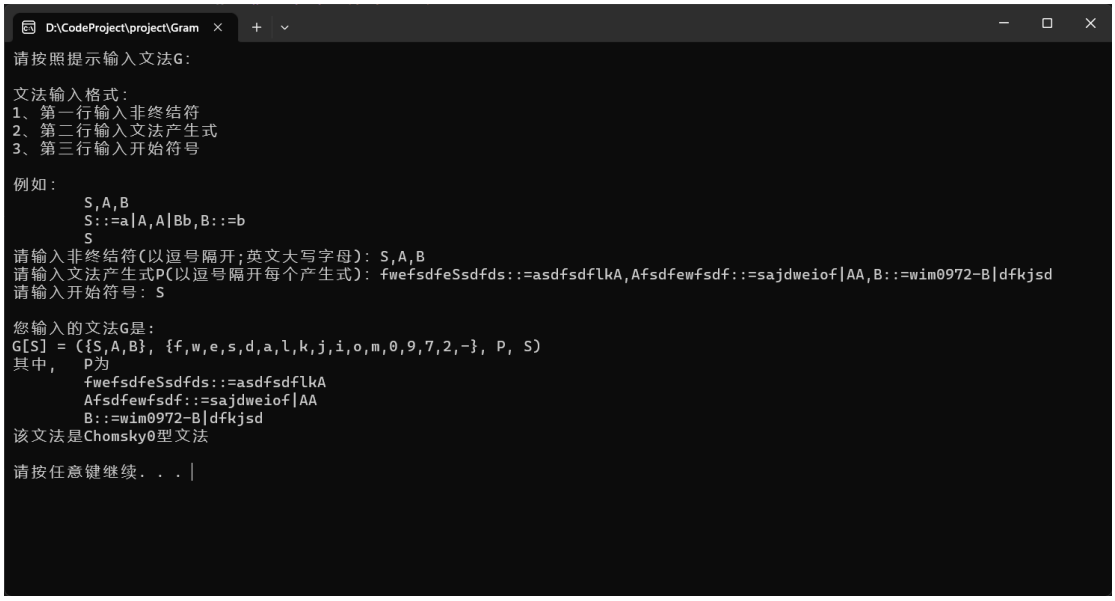


第三部分 测试结果

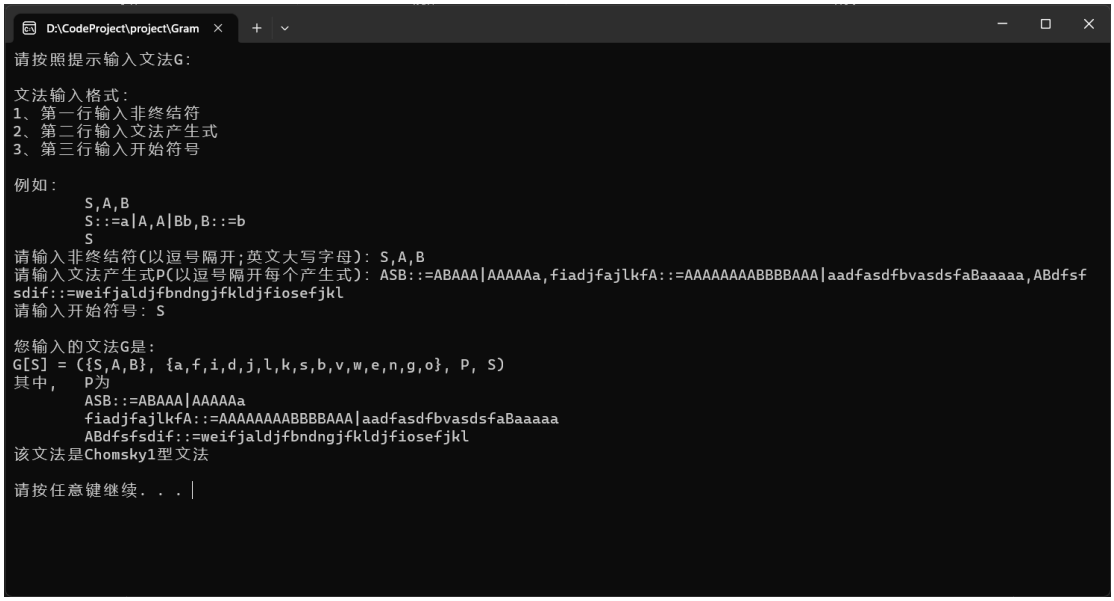
#0 测试结果如下：



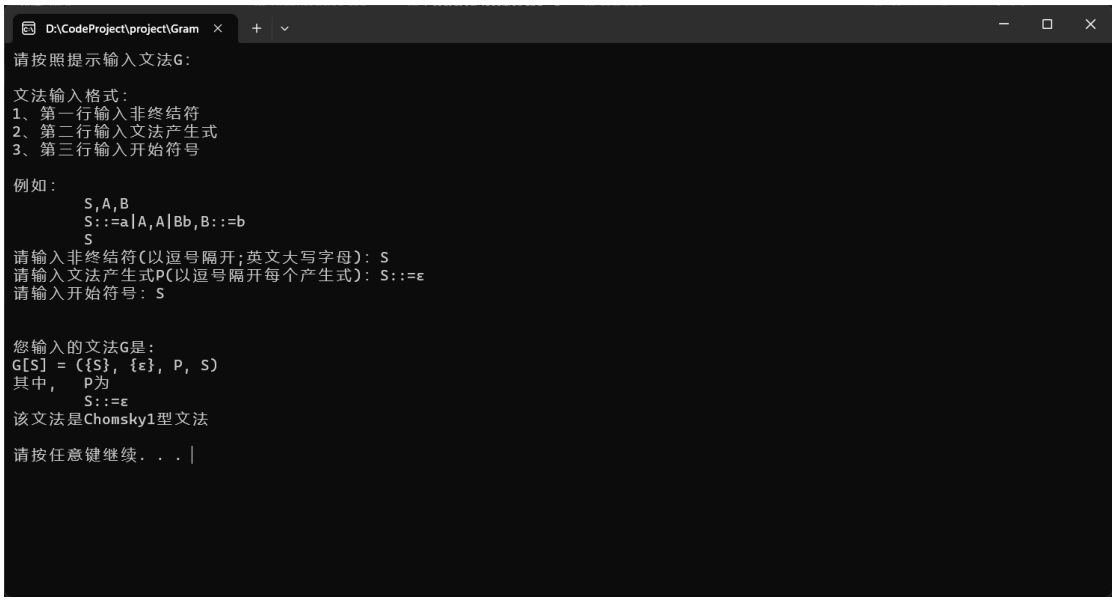
#1 测试结果如下：



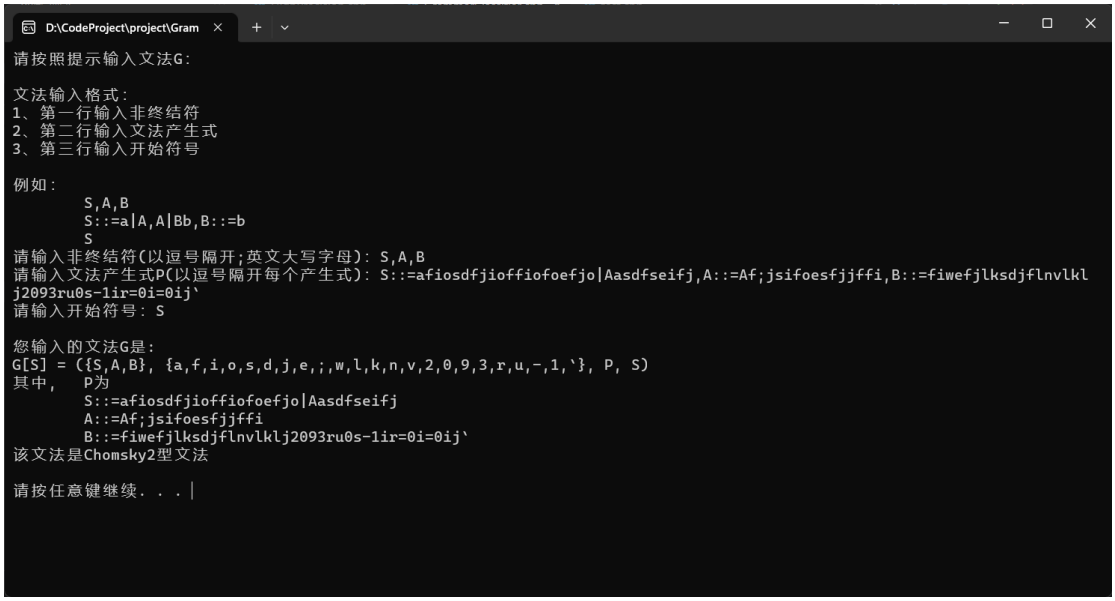
#2 测试结果如下：



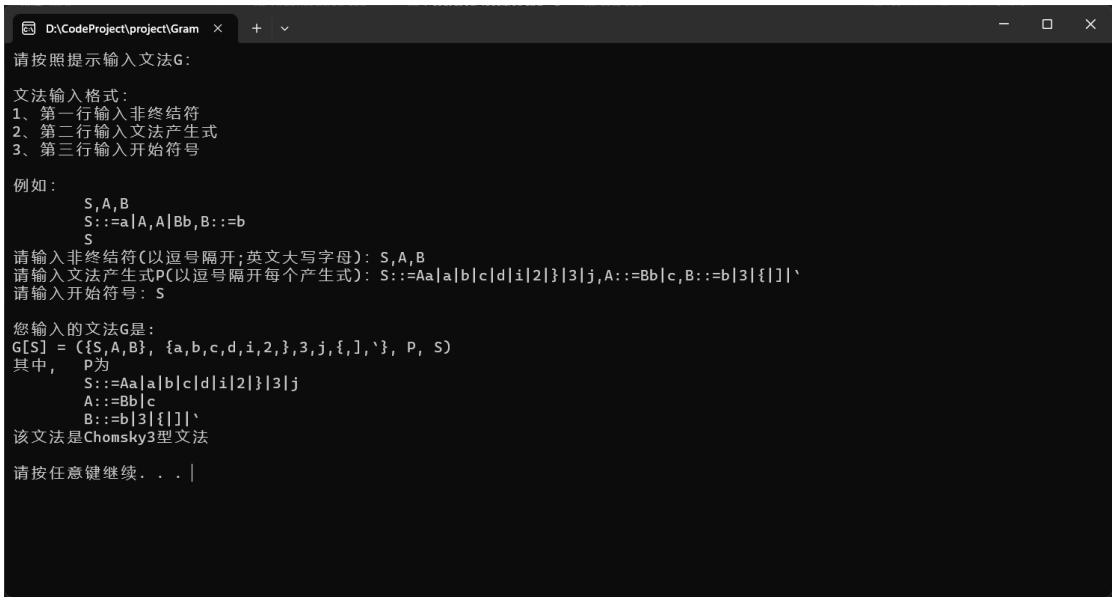
特殊情况：



#3 测试结果如下：



#4 测试结果如下：



测试结果完全正确，符合预期