深 圳 大 学 实 验 报 告

课程名称：编译原理

实验项目名称：高级语言及其文法

学院：计算机与软件学院

专业：软件工程

指导教师：蔡树彬

报告人：郑彦薇 学号：2020151022 班级：软件工程 01 班

实验时间：2023 年 2 月 28 日至 3 月 19 日

实验报告提交时间：2023 年 3 月 17 日

教务处制

实验目的与要求：

目的：通过实验，加深对文法及其分类的理解。

要求:

第一部分：文法的分类

输入：读入一个名为“g.in”，表示一个文法的文本文件。文件由 n 行组成，每行表示一个产生式，产生式是形如“S→a”的字符串，终结符由小写字母或数字表示，非终结符由大写字母表示，第一行的第一个非终结符为开始符号。

处理：对文法进行分类

输出：控制台输出，明确这是一个 0/1/2/3 型文法

第二部分：3 型文法的推导

输入：第一部分实验的输入文件，并在最后面，增加一个字符串

处理：首先判断输入文法是否 3 型文法，如果是 3 型文法，再判断该文法能否推导生成给定字符串，若可以，给出推导过程。其它情况输出自由定义（可自由尝试 0、1、2 型文法的推导情况）

输出：控制台输出，给出给定字符串的推导过程（若存在）或其他说明信息。

方法、步骤：

要完成本实验，依据实验要求进行分解，需要完成的实验步骤是：

1. 如何设计、实现（或存储、处理）文法？

特别的，如果把文法设计为一个类，那么这个类的**公有方法**应该设计为哪些？

请采用面向对象的方法，对该类的数据和方法给出详细的解释。

答：对于一个文法，根据文法概念可以知道文法中包含非终结符集、终结符集、产生式集以及开始符号。

于是对于文法的存储，可以定义一个文法类，对于该文法类，公有方法应包括：文法的构造方法 readFile()、判断是否是文法的方法 LegalJudge()、判断文法类型的方法以及文法能否推导给定字符串的判断方法 derive()。私有变量包括：用于存储大写字母的 vector、存储小写字母的 vector、存储产生式集的 vector 以及表示开始符号的 char 型变量。

把每个方法以及需要的数据列举如下：

```
//读取文件方法
void readFile()
//判断是否是合法文法
void LegalJudge()
//判断文法类型
bool FirstJudge()
bool SecondJudge()
bool ThirdJudge()
//能否推导给定字符串方法
bool derive(string nonTerminal, string input)
```

2. 如何判定一个文法是 0、1、2、3 型文法？

答：0 型文法：只要判断文法合法，则当前文法首先满足 0 型文法的要求。判定文法是否合法的方法：读取文件，根据产生式箭头位置划分产生式的左部和右部。存在以下几种情况则

不是合法文法，否则是 0 型文法。情况 1：产生式左部长度为 0 或右部长度为 0；情况 2：所有产生式的右部都包含大写字母；情况 3：产生式左侧只有小写字母。
1 型文法：首先判断文法是否是 0 型文法，接着获取每一个产生式左部的长度和右部的长度，判断是否存在产生式左部长度大于右部长度。若存在，则文法不是 1 型文法；若不存在，则左右产生式都满足 1 型文法的要求，所判断文法为 1 型文法。
2 型文法：首先判断文法是否是 1 型文法，接着获取每一个产生式的左部，判断其是否是一个非终结符。如果同时满足左部长度为 1 且对应字符为大写字母，则文法为 2 型文法，否则不是。
3 型文法：首先判断文法是否是 2 型文法，接着获取每一个产生式的右部，判断长度是否为 1 或 2。继而判断长度为 1 时，对应字符是否是终结符；长度为 2 时，对应字符是否为一个终结符和一个非终结符。若满足以上两种情况，则文法为 3 型文法，否则不是。

3. 针对 3 型文法，如何判断开始符号 S 能否推导出给定符号串 a?

答：在上题中，我们已经提出了如何对文法类型进行判断。对于给定符号串能否被文法推导在对文法进行判断的过程中，我们可以采取一种类似链表的方式，定义一个产生式列表存储每一个产生式中的非终结符及对应的产生式。推导过程使用向下递归的算法，对给定字符串的每一位进行判断，如果是非终结符，就在产生式列表中进行查找，判断是否能通过该非终结符下对应的产生式推导出对应的符号串；如果是终结符，就直接进行比较，判断是否相等。遍历每一个产生式（即对每一个可能进行判断），判断是否存在产生式能够与给定符号串匹配成功。
若已遍历所有产生式仍找不到存在匹配的项，则说明给定符号串无法通过 3 型文法进行推导。

4. 如果针对的是 2 型文法，前述 3 型文法的推导算法能否正常运行？如果不能，将可能出现什么问题？你将如何解决？

答：不可以。2 型文法包含的情况比 3 型文法更多，更加复杂。如 2 型文法的非终结符可以产生空串，可以出现多个非终结符，而不像 3 型文法只能有 a 或 aA 形式。总之，在编写 2 型文法推导方法时，需要对以上两种情况的判断。

实验过程及内容：

一、文法的分类

1. 读取文件，构造文法并判断文法的合法性

（1）根据上述所述思路，首先读取文件，按箭头位置对产生式的左右部进行拆分。这一步的目的是文法的合法性判断、类型判断都需要分别对左右部进行分析。
拆分方法：读取文件的每一行，找到箭头的位置，在字符“-”的左边即为产生式的左部；在字符“>”的右边即为产生式的右部。并以字符“-”的下标为拆分，得到左部的长度和右部的长度，得到这一部分的数据可以应用于对文法类型的判断。
（2）接着针对以下条件逐步对文法合法性进行判断：
①判断文法产生式的左右是否无符号，即是否为空。如果为空，则不合法。在编程中只需要判断上一步得到的左右部的长度是否为 0 即可。
②对左右字符进行判断，判断产生式的左部是否有大写字符（非终结符）以及右部是否都是大写字符。若左部没有大写字符，产生式错误；若每一个产生式的右部都有大写字符，则说明根据这些产生式只能不停的推导出矩形，不能得到任何句子，此时构造出来的文法也不合

法。这里使用标识符对每一行的判断情况进行记录，循环结束后对这两个值进行判断即可。下面展示这一部分的代码及相关解释。

```
    for (int i = 0; i < left_len; i++)
    {
        ch_left[i] = ch[i];
        //cout << ch_left[i] << " ";
        //左边有大写字母
        if (ch_left[i] >= 'A' && ch_left[i] <= 'Z') {
            flag = 1;
        }
    }
    //cout << endl;
    for (int i = 0, j = left_len + 2; i < right_len; i++, j++) {
        ch_right[i] = ch[j];
        //cout << ch_right[i] << " ";
        if (ch_right[i] >= 'A' && ch_right[i] <= 'Z') {
            cnt = 1; //产生式右侧有大写，cnt标识为1
        }
    }
    //cout << endl;
}
```

③判断结束后，使用 erase 方法对大小写字符（即非终结符和终结符）进行进一步的去重处理，获得非终结符列表和终结符列表，并在非终结符列表中得到文法的开始符号，最后按照文法定义的格式进行发输出，完成文法的构造。

```
    //使用vector实现数组的去重以及排序输出
    sort(temp1.begin(), temp1.end());
    temp1.erase(unique(temp1.begin(), temp1.end()), temp1.end());//去除重复元素
    cout << "所有非终结符:";
    for (auto x1 : temp1)
        cout << x1 << " ";
    cout << endl;

    sort(temp2.begin(), temp2.end());
    temp2.erase(unique(temp2.begin(), temp2.end()), temp2.end());//去除重复元素
    cout << "所有终结符:";
    for (auto x2 : temp2)
        cout << x2 << " ";
    cout << endl;
```

2. 对合法的文法进行类型判断

首先，在上述文法构造中对文法合法性的判断即完成对文法类型中 0 型文法的判断。

①根据方法、步骤中对 1 型文法的判断方法，在编程中比较产生式左右部的长度关系即可。

```
if (leftlen > rightlen) {
    cout << "存在产生式的左部长度大于右部长度，该文法不是1型文法" << endl;
    return false;
}
```

②对于 2 型文法，首先判断是否是 1 型文法。

如果是，同样按照方法、步骤中所述判断方法设置条件语句对产生式左部进行判断。设置的判断条件应包括：产生式左部的长度是否为 1；产生式左部长度为 1 时，左部是否是一个非终结符。

```
if ((leftlen == 1) && (P[i][leftlen - 1] < 'A' || P[i][leftlen - 1] > 'Z')) {
    //产生式左部是一个非终结符
    cout << "存在产生式的左部长度为1，但不是非终结符，该文法不是2型文法" << endl
    return false;
}
else if (leftlen != 1) {
    cout << "存在产生式的左部长度不为1，该文法不是2型文法" << endl;
    return false;
}
```

③对于 3 型文法，首先判断是否是 2 型文法。如果是，按照方法、步骤中所示判断方法，设置条件语句对产生式右部进行判断即可。设置的判断条件应包括：产生式右部的长度是否为 1 或 2；如果长度为 1，是否是一个终结符；如果长度为 2，是否是一个非终结符和一个终结符。

```
if (rightlen != 2 && rightlen != 1) {
    cout << "存在产生式右部长度不为1或2，该文法不是3型文法" << endl;
    return false;
}
//长度为1，判断是不是终结符
else if (rightlen == 1 && (P[i][j] < 'a' || P[i][j] > 'z')) {
    cout << "存在产生式右部长度为1，但不是终结符，该文法不是3型文法" << endl;
    return false;
}
else if (rightlen == 2) {
    if ((P[i][j - 1] < 'a' || P[i][j - 1]>'z') || (P[i][j] < 'A' || P[i][j]>'Z')) {
        cout << "存在产生式右部长度为2，但不为A->aB形式，该文法不是3型文法" << endl;
        return false;
    }
}
```

3. 运行结果

3.1. 设计一个不合法的文法对上述的构造以及合法性判断过程进行验证：

```
S->aSBE
S->aBE
EB->B
产生式右侧均有大写，无法结束，不合法!
```

3.2. 设计简单文法对上述文法类型判断过程进行验证，同时验证文法的构造是否正确。

设计文法 1 对 1 型文法的判断方法进行验证：

```
A->aB
B->bCa
C->acD
aD->c
每一行判断结束，可以构造文法
所有非终结符:A B C D
所有终结符:a b c
构造文法为：G=({A,B,C,D},{a,b,c},{A->aB,B->bCa,C->acD,aD->c},A)
存在产生式的左部长度大于右部长度，该文法不是1型文法
```

设计文法 2 对 1、2、3 型文法的判断方法进行验证：

```
A->aB
B->bCa
C->acD
D->c
每一行判断结束，可以构造文法
所有非终结符:A B C D
所有终结符:a b c
构造文法为：G=({A,B,C,D},{a,b,c},{A->aB,B->bCa,C->acD,D->c},A)
该文法是1型文法
该文法是2型文法
存在产生式右部长度不为1或2，该文法不是3型文法
```

设计文法 3 对 1、2、3 型文法的判断方法进行验证：

```
A->aB
B->bC
C->cD
D->c
每一行判断结束，可以构造文法
所有非终结符:A B C D
所有终结符:a b c
构造文法为：G=({A,B,C,D},{a,b,c},{A->aB,B->bC,C->cD,D->c},A)
该文法是1型文法
该文法是2型文法
该文法是3型文法
```

可以看到程序正确调用了文法的构造方法以及各个文法类型的判断方法，并得到正确结论。

二、3 型文法的推导

1. 读取文件，获得给定字符串，判断能否通过文法推导

①根据上述方法、步骤中所述，在编程中使用 unordered_map 数据结构对 3 型文法产生式列表进行存储，列表中的每一个表达式都包含两部分，分别为字符存放非终结符（即产生式左部）以及字符串存放产生式右部。

```
//非终结符和产生式列表
unordered_map<string, vector<string>> productions;
```

②对于产生式列表中的每一个产生式，获取其符号列表，用于进行方法、步骤中所说的“使给定符号串与每个产生式中的符号进行匹配”的步骤。

设置一个用于 string 类型的 vector 变量 symbols，用于存放从产生式中分解出来的每一个符号。在与给定符号串进行匹配时，可以使用解析过程中得到的 symbols 数据进行匹配。符号列表的获取方法如下。

```
//把符号序列解析为一个个符号
vector<string>symbols;
size_t i = 0;
while (i < production.length()) {
    //判断是否是一个终结符
    if (production[i] == '\"') {
        size_t j = i + 1;
        while (j < production.length() && production[j] != '\"') {
            j++;
        }
        symbols.push_back(production.substr(i, j - i + 1));
    }
    else {
        //否则，就是一个非终结符
        symbols.push_back(string(1, production[i]));
    }
    i++;
}
```

③接着，使用上一步得到的符号序列与给定字符串中的字符进行一一匹配。

如果存在匹配项，就判断当前匹配的符号为终结符还是非终结符，如果是非终结符，则查找是否有对应的产生式能够推导出相应字符串；如果是终结符，表示不需要进行推导，直接比较是否相等。

如果当前产生式不存在匹配项，则直接跳出当前循环，与下一个产生式的符号列表进行匹配判断。让给定的符号串与所有产生式都进行匹配的判断，若已遍历了所有产生式，仍在每一个产生式的符号列表中找不到匹配项，则说明该给定的符号串不能通过该 3 型文法进行推导。

2. 运行结果

设计文法对程序中 3 型文法的判断方法进行验证：

```
S->aA
A->aA
A->B
该文法不是3型文法，也不能进行推导
```

设计文法对程序中文法合法但不能推导给定符号串的情况进行验证：

```
S->aA
A->aA
A->b
该文法是3型文法，接下来进行推导判断
给定符号串:ba
给定符号串不能被推导出来
```

设计文法对程序中文法合法且能够推导给定符号串的情况进行验证：

```
S->aA
A->aA
A->b
该文法是3型文法，接下来进行推导判断
给定符号串:ab
给定符号串能够被推导出来
```

实验结论：

1. 文法分类的测试用例设计与说明

为了验证你第一部分编写的程序是准确的，你设计了什么测试数据（测试用例）进行测试，你**设计测试数据的要点**是什么，有什么目的？最后得到的结果如何。

答：在设计第一部分的测试数据时，为了验证文法判断是否正确，因为程序 1、2、3 型文法的判断都首先需要对上一种类型进行判断，所以测试数据可以尽量的覆盖每一种判断。如在第一部分中使用的测试用例，通过判断文法是否是 3 型文法的同时，能够对 1、2 型文法的判断方法进行引调用，同时验证了这两个判断的正确性。

2. 文法推导的测试用例设计与说明

为了验证你第二部分编写的程序是准确的，你设计了什么测试数据（测试用例）进行测试，你**设计测试数据的要点**是什么，有什么目的？最后得到的结果如何。

答：在设计第二部分的测试数据时，为验证文法判断函数以及文法推导是否正确，测试数据同样需覆盖每一种判断。如在该部分使用的测试用例，分别设计了数据对文法不合法、文法合法但无法推导给定符号串以及文法合法且能推导给定符号串这 3 种情况进行了验证。

心得体会：

1. 通过本实验，对文法的分类方法和推导过程有了更深的认识。清楚如何通过编程实现对文法合法性以及文法类型的判断；以及如何通过已知文法对给定的字符串进行推导。

2. 文法推导的过程是一个从开始符号开始，合理选择文法中的产生式，逐渐得到目标符号串的过程。在推导过程中，我们需要不断进行产生式左部与目标符号串的符号进行匹配与替换，这是一个可以利用递归方式实现的过程。学会如何利用递归算法对文法进行推导，同时清楚不同的文法类型的推导方法并不是唯一，需要根据文法产生式的类型进行修改和完善。

指导教师批阅意见：

成绩评定：

指导教师签字：蔡树彬

2023 年 3 月 15 日

备注：