|  |  |  |
| --- | --- | --- |
| **1、实验目的与要求**  **一、实验目的：**  本实验旨在通过设计和实现一个程序，对给定的文法文件进行读取和分类，加深对高级语言文法结构的理解，并掌握如何使用编程语言来实现文法的定义和分类。通过本实验，学生将能够：  ①、理解和掌握文法四元组（V,T,P,S）的定义和文法文件的组织形式  ②、设计和实现一个程序，能够读取文法文件并解析其中的文法规则；  ③、理解和掌握 Chomsky 文法体系，并能够编写程序自动判断文法的类型；  **二、实验要求：**  ①、使用 C、C++、或 Java 完成任务一、二的程序编写； | | |
| **2、实验内容**  文法（Grammar）是描述高级语言语法结构的重要工具。定义任意的文法 G ， 需要完成对其四元组（V,T,P,S）的定义（课本 P33）。在该实验中，请制定文法 文件的具体组织形式、编程完成对文法文件的读取、并完成对文法的分类。该实 验具体包含以下两个任务：  **·任务一：文法的定义及读取**  现规定文法由 Grammar.txt 文件保存，请制定文法文件的具体存储格式。如文法 G={{S,A,B,C}, {a,b,c}, {S->ABC,A->a,B->b,C->c}, S}在 Grammar.txt 文件中可由以下方式描述并存储：    文法的文本形式可根据自己需要自由定义，在此基础上， 编程实现对任意文 法文件的读取。  **·任务二：文法的分类**  根据 Chomsky 的文法体系分类（课本 P40），文法分为四大种类。请在任务 一的基础上， 编程实现对 Grammar.txt 中存储的文法进行分类， 自动判断其所属 类别。例如任务一中所给出的文法 G 应被判定为 2 型文法， 即上下文无关文法。 请设计分类方法，并设计四类不同的测试文法测试分类结果的正确性。 | | |
| **3、实验过程步骤及说明**  本次实验使用的是Java语言完成任务一、二的程序编写。  **一、任务一：文法的定义及读取**  G={{S,A,B,C}, {a,b,c}, {S->ABC,A->a,B->b,C->c}, S}文法由 Grammar.txt文件保存。如下图所示：    图 1 文法的初始保存格式  **1、Grammar类的设计**  首先编写一个Grammar类，包含五个类成员变量，用于表示文法的四个结构V、T、P、S，即文法的非终结符号集合、终结符号集合、产生式集合和开始符号。  如下图所示，其中产生式用两个集合存储，一个存储产生式左部、一个存储产生式右部，以便实现任务二中对文法的分类。    图 2 Grammar类的成员变量  2、文法格式转换  为了更好地读取文法各部分内容，先设计一个convertGrammarToFile函数，读取原始文法文件的每一行，并将每一行的内容转换成适合程序处理的格式。转换后的文法如下图所示：    图 3 便于读取的文法存储格式  下面介绍具体的实现过程：  ①、根据源文件的路径读取文法数据并存储在data列表中  convertGrammarToFile函数包含两个参数，inputFilePath——原始文法文件的路径，outputFilePath——输出文件的路径。首先根据源文件的路径读取文法数据并存储在data列表中，如下图所示。    图 4 根据源文件的路径读取文法数据并存储在data列表中  **②、将转换后的文法内容写入新文件**  **A、截取最外层大括号内的内容**  使用循环遍历文法数据列表data中的每一行文法规则。对于每一行文法规则，创建一个长度为4的字符串数组Grammar，用于存储文法规则的四个部分。然后获取当前行的文法规则，并找到左花括号的位置pos，接着使用substring方法截取左右花括号之间的内容，并赋值给center。  以G={{S,A,B,C}, {a,b,c}, {S->ABC,A->a,B->b,C->c}, S}为例子，截取的内容为{S,A,B,C}, {a,b,c}, {S->ABC,A->a,B->b,C->c}, S。    图 5 截取大括号内的内容  **B、分割V、T、P、S四个部分内容**  对于当前行的文法规则，使用循环遍历其中的四个部分。  在每一次循环中，查找右花括号的位置pos，接着进行判断：  · 如果找不到右花括号（即开始符号），则将剩余部分作为最后一个部分，并加上换行符，然后将该部分写入文件。  · 如果找到了右花括号，则截取当前部分并加上换行符，并更新center，去掉已经处理的部分及其后的两个字符，然后将该部分写入文件。  对应代码如下图所示：    图 6 分割V、T、P、S四个部分内容  **C、文法格式转换测试**  在主方法中，创建一个Grammar实例，调用convertGrammarToFile函数进程文法格式的转换，代码如下图所示：    图 7 文法格式转换测试    图 8 源文件    图 9 转换后文件  **3、文法内容读取**  对转换后的文件内容进行读取，进一步分割出文法四个部分的内容。  **①、读取转换后文件的内容**  创建一个空的字符串列表myFile，用于存储从文件中读取的文本行。通过BufferedReader逐行读取文件内容，直到文件末尾，每读取一行文本内容，将其添加到myFile列表中。最后调用extractGrammarData(myFile)方法，将读取的文法数据传递给该方法进行进一步处理。  代码如下图所示：    图 10 读取转换后文件的内容到myFile列表  **②、具体分割文法各个部分的内容**  根据转换后的文法格式，可知V、T、P、S分别对应文件中第一、二、三、四行的内容。下面介绍具体的分割方式：  **A、提取V（非终结符号集合）、T（终结符号集合）**  提取V、T元素的思路类似，因为他们的元素之间都用逗号分割开，因此**只需要遍历该行内容，逐个字符判断是否为逗号，如果不是逗号，则将其添加到对应集合中。**其中非终结符号对应第一行的内容，终结符号对应第二行的内容。  代码如下图所示：    图 11 提取V（非终结符号集合）、T（终结符号集合）的代码  **B、提取产生式的左部集合（P1）和右部集合(P2)**  为了便于后续对文法的分类判断，这里需要将产生式的左右部分离开来，即考虑右部有候选式的情况。  首先从myFile列表中获取第三行文本内容，即产生式集合。然后使用循环处理该行内容，每次处理一个产生式，直至该行为空，。  在每次循环中，**首先查找逗号首次出现的位置，以分割当前需要处理的产生式**。对于当前产生式，**检查是否存在箭头符号（"->"），如果存在，则将产生式左部和右部分别提取出来。如果右部存在多个候选项（用竖线"|"分隔），则将其分割并分别存储。**  最后，将每个左部和右部对应的产生式添加到P1和P2列表中。  代码如下图所示    图 12 提取产生式的左部集合（P1）和右部集合(P2)  **C、提取开始符号S**  从myFile列表中获取第四行文本内容，即开始符号。因为开始符号只有一个，所以提取第一个字符即可。代码如下图所示：    图 13 提取开始符号S  **D、打印文法的各个部分内容**  编写一个printGrammar方法，输出文法各个部分内容。其中产生式需要用“->”符号将左右部组合起来。对应的代码如下图所示：    图 14 打印文法的各个部分内容  E、读取文法内容测试：  主方法在转换文法格式的基础上，调用readGrammarFromFile函数读取转换后的文法数据，再调用printGrammar函数打印文法各个部分的内容，代码如下图所示：    图 15 读取文法内容测试代码  输出结果如下图所示，可以正确实现对文法文件的读取。    图 16 读取文法后的输出结果  综上，通过文法格式转换、读取转换后文件并提取文法各个部分内容，成功实现对文法文件的读取。同时，如果文法直接以转换后的形式存储，那么可以直接调用读取方法进行读取，不需要进行转换。因此通过上述程序，**实际上实现了对两种存储方式的文法的读取。**  **二、任务二：文法的分类**  1、编写判断文法类型的代码：  我编写了一个checkGrammarType方法用于确认某个文法具体属于哪种类型。    图 17 checkGrammarType函数代码片段  如上图所示，首先定义四个变量，其中one\_of\_type用于表示当前产生式符合的文法类型，type表示文法类型，这两个变量都初始为4表示未确认。is\_left和is\_right变量用于3型文法的判断，检测是否出现左右线性文法混用的情况。  然后循环遍历每个产生式，进行文法的判断。其中 P1 存储产生式左部，P2 存储产生式右部。每次循环初始需要更新文法类型，其中如果one\_of\_type小于type，则更新type。    图 18 使用循环遍历产生式列表  **·、0型文法：**  定义：设G=（V，T，P，S），如果它的每个产生式α→β是这样一种结构：**α∈且至少含有一个非终结符，而 β∈(V∪T)\*，**则G是一个0型文法。  0型文法也称短语文法。一个非常重要的理论结果是：0型文法的能力相当于图灵机(Turing)。或者说，任 何0型文语言都是递归可枚举的，反之，递归可枚举集必定是一个0型语言。0型文法是这几类文法中，限制最少的一个。  根据0型文法的定义，首先检查左部α是否只包含终结符和非终结符，若有非法符号则标记为无效。代码如下图所示：    图 19 检查左部α是否只包含终结符和非终结符  接着判断左部α是否至少含有一个非终结符，若有则更新标记符号has\_Vp为true。代码如下图所示：    图 20 判断左部α是否至少含有一个非终结符  检查右部β是否只包含终结符和非终结符，若有非法符号则标记为无效。这里需要考虑右部含有空串的情况，空串不属于非法符号。    图 21 检查右部符号是否有效  最后根据以上三次判断，如果有一个不符合0型文法的定义，则不符合0型文法，退出整个循环，更新type为-1。如果符合0型文法的定义，则将one\_of\_type更新为0。    图 22 综合完成0型文法的判断  **· 1型文法：**  1型文法也叫上下文有关文法，此文法对应于线性有界自动机。  它是**在0型文法的基础上每一个α→β,都有|β|>=|α|（α→ε除外）。**这里的|β|表示的是β的长度。  如有A->Ba则|β|=2,|α|=1符合1型文法要求。反之,如aA->a，则不符合1型文法。  根据1型文法的定义，首先计算右部除去空串ε的长度，如果左部α的长度小于等于右部β的长度或者β为空串，则将 one\_of\_type 设置为1型文法。如果不符合要求，则结束本次循环，不进行2型、3型文法的判断，直接对下一产生式进行判断。    图 23 1型文法的判断  **· 2型文法：**  2型文法也叫上下文无关文法，它对应于下推自动机。  如果对于任意α→β∈P，均有|β|≥|α|，并且α∈V成立，则称G为２型文法  即2型文法是在1型文法的基础上,再满足：每一个α→β都有α是非终结符。  如A->Ba,符合2型文法要求。而Ab->Bab虽然符合1型文法要求,但不符合2型文法要求，因为其α=Ab，而Ab不是一个非终结符。  判断2型文法的代码如下图所示，只需要在1型文法的基础上，检查左部α是否只有一个非终结符，若符合则将 one\_of\_type 设置为2型文法。    图 24 2型文法的的判断  ·3型文法  3型文法也叫正规文法，它对应于有限状态自动机。  它是在2型文法的基础上满足:A→α|αB（右线性）或A→α|Bα（左线性），其中A、B∈V，α∈T或为空串。  这里需要特别注意的是左、右线性文法不可混用。  根据3型文法的定义，判断：  若右部β只有一个非终结符或者只有空串，则将 one\_of\_type 设置为3型文法。  若右部β有两个符号，并且一个是非终结符，一个是终结符或空串，则判断是左线性文法还是右线性文法，并标记对应的标志位。  其中如果循环过程中出现左右线性标志位同时为1的情况，说明出现了左右线性文法混用，不符合3型文法的定义。所以只有当其中一个线性标志位为1另一个线性标志位为0的情况才符合3型文法的定义。代码如下图所示：    图 25 3型文法的判断  循环遍历每个产生式结束后，再次根据one\_of\_type和type的大小关系更新type并返回。    图 26 最后更新并返回文法类型  最后编写一个analyzeGrammarType方法输出文法类型。它调用checkGrammarType方法获取文法的类型type，然后根据type输出文法的类型。其中如果type是-1，则表明文法不是正规文法。    图 27 输出文法的类型  **2、测试程序能否正确判断文法类型**  ①、特殊情况测试（右部有多项和空产生式）  A、右部为候选式  如下图所示，测试样例右部为候选式，测试对该文件的读取和文法类型判断。    图 28 右部为候选式测试    图 29 右部为候选式测试结果  结果上图所示，可以正确提取所有产生式的左部和右部，对于候选式，将拆分为多个产生式，如B->b|c拆分为B->b和B->c。同时可以正确判断文法的类型。  **B、测试空产生式**  如下图所示，测试样例中有空产生式，测试对该文件的读取和文法类型判断。    图 30 测试空产生式  结果如下图所示，可以正确提取所有产生式的左部和右部并正确判断文法类型。    图 31 测试空产生式结果  **②、测试0型文法**  如下图所示，G={{S,A,B,C}, {a,b,c}, {S->ABC|ε,AB->a,B->b|c,C->c}, S}为一个0型文法，因为AB->a不符合1型文法的定义。    图 32 0型文法样例  测试结果如下，可以正确提取所有产生式的左部和右部并正确判断文法类型。    图 33 0型文法样例测试结果  **③、测试1型文法**  如下图所示，G={{S,A,B,C}, {a,b,c}, { S->ABC|ε,A->a|AB,AB->ab,B->b|c,C->c}, S}, S}为一个1型文法，因为AB->ab不符合2型文法的定义。    图 34 1型文法样例  测试结果如下，可以正确提取所有产生式的左部和右部并正确判断文法类型。    图 35 1型文法样例测试结果  **④、测试2型文法**  2型文法使用的样例是G={{S,A,B,C}, {a,b,c}, {S->ABC,A->a,B->b,C->c}, S}，该文法符合2型文法的定义，但是S->ABC不符合3型文法的定义。    图 36 2型文法测试样例  测试结果如下，可以正确提取所有产生式的左部和右部并正确判断文法类型。    图 37 2型文法样例测试结果  **⑤、测试3型文法**  3型文法测试样例为G={{S,A,B,C}, {a,b,c}, {S->aA|bB|cC,A->a,B->b,C->c}, S}，它符合3型文法的所有要求。    图 38 3型文法测试样例  测试结果如下，可以正确提取所有产生式的左部和右部并正确判断文法类型。    图 39 3型文法样例测试结果  **⑥、左右线性文法混用情况测试**  根据3型文法的定义，不可以混用左、右线性文法。混用时只符合2型文法的定义。例如G={{S,A,B,C}, {a,b,c}, {S->aA|bB|Cc,A->a,B->b,C->c}, S}，其中S->aA和S->bB属于右线性文法，而S->Cc属于左线性文法。    图 40 左右线性文法混用样例测试  结果如下图所示，可以正确判断左、右线性文法混用的情况。    图 41 左右线性文法混用样例测试结果 |
| **4、实验总结：**  在本次实验中，我设计并实现了一个程序，用于读取给定的文法文件并对其进行分类。首先定义文法的四元组（V，T，P，S），然后编写了程序来解析给定的文法文件，提取其中的文法规则。接着，我还实现了文法分类的功能，根据 Chomsky 的文法体系将文法分为四种类型：0型、1型、2型和3型。通过逐个检查文法规则的特征，我设计的程序能够准确地确定给定文法的类型。其中，**我还考虑到了空产生式的情况以产生式右部为候选式的情况，**充分考虑了各种可能性，成功实现对任意文法的读取和分类。  最后，我进行了一系列的测试，充分验证了程序的正确性和可靠性。综上，我很好地完成了任务一和任务二的要求。 |

深圳大学学生实验报告用纸

|  |
| --- |
| **5、实验心得体会**  通过本次实验，我收获了以下几点经验和体会：  **①、对文法结构的理解加深：**通过设计和实现程序，我更深入地理解了文法的结构和分类，包括非终结符、终结符、产生式等概念，以及 Chomsky 的文法体系。  **②、编程能力的提升：**通过编写程序来解析文法文件并进行分类判断，我加强了编程能力，特别是对文件操作、字符串处理、逻辑判断等方面的掌握。  **③、对文法分类算法的理解：**通过实现文法分类算法，我深入了解了不同类型文法的特征和区别，进一步提高了对编译原理相关知识的理解和掌握。  **④、本次实验遇到的困难与解决：**进行编程实现文法分类过程中，如何处理空产生式是一个重难点，很考验逻辑性，在不断调试思考后，我成功完成了对各种情况的处理，使得程序能够准确识别各种类型的文法。  **⑤、巩固文法基础知识：**在上学期我已经学习了《形式语言与自动机》这个课程，课程中已经学习过Chomsky的文法体系等基本概念，在本次实验中我编程实现了文法的识别和分类程序，在这个过程中，我巩固了已学知识，对文法相关知识概念有了更加深入地理解和掌握。 |