预处理与词法分析器

一、 预处理与词法分析器实现目标(也即最终实现的功能)

1. **尽量保持和框架接口的一致性：**
   在词法分析通过时，生成的 xml 结构与框架标准一致，type 标签使用数字定义，与原模块定义不同，因缺少原模块代码，无法得知完整单词 type 定义，且要自己实现语法分析和中间代码生成部分，只需最后保持四元组接口一致即可。
   在词法分析不通过时，xml 中多加入了 typeDescription 标签（描述单词类型），wrongInf 标签（描述该单词的错误信息），src 标签（描述错误所在源文件）
2. **使用方法如下：**
   将 cplScanner.java 导入项目中
   禁用预处理部分，在 scan 部分创建 cplScanner sc=new cplScanner();
   调用 sc.run(cFile, scOutFile);方法执行预处理和词法分析部分，**cFile 应为 xx\xx.c(即.c 源文件),scOutFile 应为 xx\xx.token.xml**
   可沿用框架中的代码产生 cFile 和 scOutFile，不过生成 scOutFile 时的代码应变动如下：String scOutFile = cFile.replace(MiniCCCfg.MINICC_PP_INPUT_EXT, MiniCCCfg.MINICC_SCANNER_OUTPUT_EXT);
3. **保留正确的行信息**以供后续分析报错使用
4. 生成文件清单：
   .pp1.c 除去单行注释结果
   .pp2.c 除去所有注释结果
   .pp3.c 完成 include 替换结果
   [.pp1.h .pp2.h .pp3.h](若有#include “xx.h”且文件存在，则会生成)
   .token.xml 词法分析结果
5. **支持的 c 语言特性：**
   1) 预处理功能:
      a) 支持**宏替换和宏定义分析**。
         错误的宏会报错（重定义的宏，缺少标识符等），并在分词结果的输出流（xml）中保留错误信息。
         正确的宏会进行替换，宏不会作为单词保留在分词的输出流中。
         宏替换会替换宏定义下方所有出现的非宏定义标识符。即支持 VS 的如下特性，若代码如下：
         #define A 10
         #define b A
         int t=b;
         则 b 被替换为 10；但是#define 后的标识符不会被替换，若下方再有一个#define A，则会报重定义错误

      b) **支持 c89 标准全部的注释特性**：//，/**/。支持去除跨行注释，能够正确识别嵌套注释如/**///*/*/*//等
      c) **include 文件包含**：支持#include “xx“，支持嵌套包含。不支持#include <xx>，包含文件必须与.c 文件在同一目录下。若有不存在的 include 文件会直接报错并跳过后续分析过程。

2) 词法分析功能：

a) 支持识别的**数字常量**：

十进制：整型和浮点型

八进制:以 0 开头

十六进制:0x 中的 x 必须小写

b) 支持识别**标识符**

标识符规则：字母或下划线开头，由字母数字下划线构成，不与关键字冲突

c) 支持识别**字符常量字符串常量**的合法性

是不是完整的字符串，字符串中是否还有非法的转义序列

d) 支持识别 **C 语言全部的转义序列**：

转义字符	意义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT)（跳到下一个TAB位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符"\'	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034
\?	代表一个问号	063
\0	空字符(NULL)	000
\ddd	1到3位八进制数所代表的任意字符	三位八进制
\xhh	1到2位十六进制所代表的任意字符	二位十六进制

e) 支持识别 c 语言 c89 标准全部的**关键字**：

关键字	说明
auto	声明自动变量
short	声明短整型变量或函数
int	声明整型变量或函数
long	声明长整型变量或函数
float	声明浮点型变量或函数
double	声明双精度变量或函数
char	声明字符型变量或函数
struct	声明结构体变量或函数
union	声明共用数据类型
enum	声明枚举类型
typedef	用以给数据类型取别名
const	声明只读变量
unsigned	声明无符号类型变量或函数
signed	声明有符号类型变量或函数
extern	声明变量是在其他文件正声明
register	声明寄存器变量
static	声明静态变量
volatile	说明变量在程序执行中可被隐含地改变
void	声明函数无返回值或无参数，声明无类型指针
if	条件语句
else	条件语句否定分支（与 if 连用）
switch	用于开关语句
case	开关语句分支
for	一种循环语句
do	循环语句的循环体
while	循环语句的循环条件
goto	无条件跳转语句
continue	结束当前循环，开始下一轮循环
break	跳出当前循环
default	开关语句中的“其他”分支
sizeof	计算数据类型长度
return	子程序返回语句（可以带参数，也可不带参数）循环条件

f) 支持识别的**运算符分隔符**：

以下截取自代码中的定义，从下面代码段中可以看到支持的运算符分隔符及他们的 type 定义（int 型）和语言描述

```
operatorMap.put("+", new typeDescription("加",33));
operatorMap.put("-", new typeDescription("减",34));
operatorMap.put("*", new typeDescription("乘/取指针内容",35));
operatorMap.put("/", new typeDescription("除",36));
operatorMap.put("++", new typeDescription("自增",37));
operatorMap.put("--", new typeDescription("自减",38));
operatorMap.put("+=", new typeDescription("加赋值",39));
operatorMap.put("*=", new typeDescription("乘赋值",40));
operatorMap.put("/=", new typeDescription("除赋值",41));
operatorMap.put("-=", new typeDescription("减赋值",42));
operatorMap.put("%", new typeDescription("模",43));
operatorMap.put("%=", new typeDescription("模赋值",44));
operatorMap.put("&", new typeDescription("位与/取地址",45));
operatorMap.put("|", new typeDescription("位或",46));
operatorMap.put("!", new typeDescription("非",47));
operatorMap.put("&&", new typeDescription("与",48));
operatorMap.put("||", new typeDescription("或",49));
operatorMap.put("->", new typeDescription("取成员",50));
operatorMap.put(".", new typeDescription("取成员",51));
operatorMap.put(">", new typeDescription("大于",52));
operatorMap.put("<", new typeDescription("小于",53));
operatorMap.put(">>", new typeDescription("右移",54));
operatorMap.put("<<", new typeDescription("左移",55));
operatorMap.put(">=", new typeDescription("大于等于",56));
operatorMap.put("<=", new typeDescription("小于等于",57));
operatorMap.put("?", new typeDescription("?:算符",58));
operatorMap.put(":", new typeDescription("?:算符",59));
operatorMap.put("!=", new typeDescription("不等于",60));
operatorMap.put("==", new typeDescription("等于判断",61));
operatorMap.put(",", new typeDescription("逗号算符",62));
operatorMap.put("(", new typeDescription("括号 s",63));
operatorMap.put(")", new typeDescription("括号 e",64));
operatorMap.put("[", new typeDescription("取数组成员 s",65));
operatorMap.put("]", new typeDescription("取数组成员 e",66));
operatorMap.put("~", new typeDescription("位非",67));
operatorMap.put("^", new typeDescription("位异或",68));
operatorMap.put("&=", new typeDescription("位与赋值",69));
operatorMap.put("|=", new typeDescription("位或赋值",70));
operatorMap.put("^=", new typeDescription("位异或赋值",71));
operatorMap.put(">>=", new typeDescription("右移赋值",72));
```

```
operatorMap.put("<<=", new typeDescription("左移赋值",73));
operatorMap.put(";", new typeDescription("语句分隔符",74));
operatorMap.put("=", new typeDescription("赋值运算符",75));
operatorMap.put("{", new typeDescription("花括号 s",76));
operatorMap.put("}", new typeDescription("花括号 e",77));
```

二、 程序的实现

1. 预处理部分：
   先去除同行内注释，产生.pp1.c 文件
   然后以.pp1.c 为输入，扫描源程序，将所有合法/**/对加入队列（先找到一个/*，再从此位置往后找*/，再找/*依此类推），并记录他们对应出现的行数。然后按队列顺序将他们替换成空串，并根据记录的行数加上对应的”\r\n“以保证代码行信息不改变，输出为.pp2.c
   然后以.pp2.c 为输入，分析文件包含，使用 hashmap 记录包含的文件避免重复 include 和递归 include 时进入死循环，分析完后产生 include 信息的 list，用于词法分析后确定行信息及源文件，输出为.pp3.c

   宏替换的功能属于预处理部分，但这里是将其和词法分析结合实现的。
2. 词法分析部分:
   1) 由一个总 DFA 实现分词和识别部分单词：
      通过一遍扫描，该 DFA 将正确划分单词并将单词初步划分为:分隔符，运算符，标识符或数字（这是一类），字符串常量，字符常量，非法的 ANSI 字符，非法的 unicode 字符，非法位置的转义字符，未经识别的结尾部分 9 大类。
   2) 当总 DFA 一次扫描完成后，遍历得到的单词集，每一类有其对应的处理子程序继续处理。
      分隔符和运算符使用统一的函数识别其类型
      标识符或数字将由处理子程序检查它是否是关键字，是否是整数，浮点数，八进制数，16 进制数以及是否符合标识符命名规则来为其分类和判断合法性。
      对于分隔符，运算符，关键字的判定是借助 hashmap 词典完成的
      对于数值常量的检查由正则表达式完成
      字符串常量的合法性由正则表达式识别
      字符常量的合法性由处理子程序和部分正则表达式识别
      未经识别的结尾将由所有类别的处理子程序处理一遍，若识别为某一类型的合法单词则完成检测，若对于所有类别都不合法则被划分为不合法类型的结尾
      对于非法的 ANSI 字符，非法的 unicode 字符，非法位置的转义字符这三类将直接添加错误描述信息，总 DFA 已经确定了错误类型
   3) 遍历完单词集后使用另一个 DFA 完成处理宏定义的操作。
      DFA 以单词流为输入，识别到合法的宏定义后会对后面的所有非 define 后标识符进行替换，然后将该宏加入宏字典，并在单词流中删除该宏。每次出现新的宏后都要判断该宏是否已被定义，若被定义要报重定义错误。对于一个非法宏，如重定义，define 后没有合法的标识符等，单词流中将保留该不合法宏及其错误信息以供出错提示和处理。

   4) 遍历单词流，根据 list 中记录的 include 文件的偏移量重定位每个单词对应的源

文件信息和行信息

这样上述步骤全部执行完后就会得到完成宏替换的单词流。若词法分析通过，单词流中将不含#define xx xx 语句。

三、 程序使用的主要方法（技术）

1. 使用二次扫描完成所有注释的删除
2. 使用 DFA 分词
3. 使用 Hashmap 建立词典查询关键字，运算符，分隔符
4. 使用正则表达式匹配标识符，数值常量，字符串常量，字符常量等
5. 使用 DFA 处理宏定义
6. 使用 HashMap 建立定义各种查询表
7. 使用 ArrayList 建立程序中各种所需队列

四、 实验结果

使用 input 目录下的 test.c 文件进行测试

test.c 内容修改如下：

```
#include "mytest.h"
int main(int a, int b){      //main function
    int p=B;
    /*
    return a + b + A;
    */
}
```

mytest.h 内容如下：

```
#define A 10
#define B A
```

生成的.pp3.c 如下：

```
#define A 10
#define B A


int main(int a, int b){
    int p=B;



}
```

执行后得到的 xml 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="test.l">
  <tokens>
```

<token>
<number>8</number>
<value>int</value>
<type>17</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>9</number>
<value>main</value>
<type>10004</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>10</number>
<value>(</value>
<type>63</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>11</number>
<value>int</value>
<type>17</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>12</number>
<value>a</value>
<type>10004</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>13</number>
<value>,</value>
<type>62</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>14</number>

<value>int</value>
<type>17</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>15</number>
<value>b</value>
<type>10004</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>16</number>
<value>)</value>
<type>64</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>17</number>
<value>{</value>
<type>76</type>
<line>3</line>
<valid>true</valid>
</token>
<token>
<number>18</number>
<value>int</value>
<type>17</type>
<line>4</line>
<valid>true</valid>
</token>
<token>
<number>19</number>
<value>p</value>
<type>10004</type>
<line>4</line>
<valid>true</valid>
</token>
<token>
<number>20</number>
<value>=</value>
<type>75</type>

```
            <line>4</line>
            <valid>true</valid>
        </token>
        <token>
            <number>3</number>
            <value>10</value>
            <type>10000</type>
            <line>4</line>
            <valid>true</valid>
        </token>
        <token>
            <number>22</number>
            <value>;</value>
            <type>74</type>
            <line>4</line>
            <valid>true</valid>
        </token>
        <token>
            <number>23</number>
            <value>}</value>
            <type>77</type>
            <line>8</line>
            <valid>true</valid>
        </token>
    </tokens>
</project>
```

从上面的结果可以看到，num 记录是从 8 开始的，说明前面有 8 个宏定义相关的符号被删掉了。程序并没有将最终的单词从 0 开始编号。后续语法分析的设计也不依据此标号，而是由从 xml 文件读取的顺序确定单词是单词流中的第几个。

将 mytest.h 修改如下：

```
#define A 10
#define B A
#define A
```

执行后得到的 xml 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="test.l">
  <tokens>
    <token>
      <number>8</number>
      <value>#</value>
      <type>100</type>
      <line>3</line>
      <valid>true</valid>
```

```
    <typeDescription>预编译符号</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\mytest.h</src>
</token>
<token>
    <number>9</number>
    <value>define</value>
    <type>101</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>宏定义标识符</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\mytest.h</src>
</token>
<token>
    <number>10</number>
    <value>A</value>
    <type>10004</type>
    <line>3</line>
    <valid>false</valid>
    <typeDescription>identifier</typeDescription>
    <wrongInf>重定义的宏</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\mytest.h</src>
</token>
<token>
    <number>11</number>
    <value>int</value>
    <type>17</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>kw17</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
    <number>12</number>
    <value>main</value>
    <type>10004</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>identifier</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
```

<token>
<number>13</number>
<value>(</value>
<type>63</type>
<line>3</line>
<valid>true</valid>
<typeDescription>括号 s</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>14</number>
<value>int</value>
<type>17</type>
<line>3</line>
<valid>true</valid>
<typeDescription>kw17</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>15</number>
<value>a</value>
<type>10004</type>
<line>3</line>
<valid>true</valid>
<typeDescription>identifier</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>16</number>
<value>,</value>
<type>62</type>
<line>3</line>
<valid>true</valid>
<typeDescription>逗号算符</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>17</number>
<value>int</value>
<type>17</type>

```
    <line>3</line>
    <valid>true</valid>
    <typeDescription>kw17</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
    <number>18</number>
    <value>b</value>
    <type>10004</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>identifier</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
    <number>19</number>
    <value>)</value>
    <type>64</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>括号 e</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
    <number>20</number>
    <value>{</value>
    <type>76</type>
    <line>3</line>
    <valid>true</valid>
    <typeDescription>花括号 s</typeDescription>
    <wrongInf>无</wrongInf>
    <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
    <number>21</number>
    <value>int</value>
    <type>17</type>
    <line>4</line>
    <valid>true</valid>
    <typeDescription>kw17</typeDescription>
    <wrongInf>无</wrongInf>
```

<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>22</number>
<value>p</value>
<type>10004</type>
<line>4</line>
<valid>true</valid>
<typeDescription>identifier</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>23</number>
<value>=</value>
<type>75</type>
<line>4</line>
<valid>true</valid>
<typeDescription>赋值运算符</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>3</number>
<value>10</value>
<type>10000</type>
<line>4</line>
<valid>true</valid>
<typeDescription>#define B</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>25</number>
<value>;</value>
<type>74</type>
<line>4</line>
<valid>true</valid>
<typeDescription>语句分隔符</typeDescription>
<wrongInf>无</wrongInf>
<src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
</token>
<token>
<number>26</number>

```
        <value>}</value>
        <type>77</type>
        <line>8</line>
        <valid>true</valid>
        <typeDescription>花括号 e</typeDescription>
        <wrongInf>无</wrongInf>
        <src>D:\工程项目\java\BIT-MiniCC-Clean\input\test.c</src>
      </token>
    </tokens>
</project>
```


这是词法分析不通过时的结果，我们可以看到报错信息为重定义的宏，错误行数以及它所在的源文件为 mytest.h

限于文章篇幅，关于更多的“一. 预处理与词法分析器实现目标”中提到的所支持的特性，请自行测试。

五、 心得体会和收获

通过此次实验使我对 c 语言的一些特性更加了解，如宏替换的规则，inlcude 规则，数值常量的定义规则，转义表等。通过自己设计 DFA 加深了对 DFA，正则文法的理解。熟悉了 java 中正则表达式的使用和 xml 的读写。