对从教学平台获取的文法：

1）撰写一份解读文档，要求对文法中每条规则所定义的语法成分进行分析，说明其作用、限定条件，给出句子示例。文件命名为"学号\_姓名\_文法解读.docx"；

2）给出一个完整的符合该文法的测试程序，要覆盖所有语法成份，代码行不少于规则的条数，并有函数或过程的递归调用。文件命名为"学号\_test.txt"；

3）为给出的测试程序给出测试用例及预期结果，要求覆盖所有分支。文件命名为"学号\_result.txt”。

请于第9周周日(11.19)24:00前提交文法解读作业，要求将作业包含的3个文件打包成一个zip文件，以“学号.zip”命名再上传。

**＜加法运算符＞ ::= +｜-**

作用：定义了加法运算符包含的符号

限定条件：注意加法运算符包括’+’和’-‘

句子示例：① 1 + 2

② 2 - 1

**＜乘法运算符＞ ::= \*｜/**

作用：定义了乘法运算符包含的符号

限定条件：注意乘法运算符包括’\*’和’/’

句子示例：① 9 \* 3

② 9 / 3

**＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==**

作用：定义了一系列关系运算符的符号

限定条件：无

句子示例：① a < b

② a <= b

③ a > b

④ a >= b

⑤ a != b

⑥ a == b

**＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z**

作用：定义了字母

限定条件：字母除了包含大小写的26个英文字母以外，还包括下划线(’\_’)

句子示例：① \_

② a

**＜数字＞ ::= ０｜＜非零数字＞**

作用：定义了数字的形式，数字可以是0或非零数字

限定条件：结合非零数字的文法可知数字的长度等于且一定等于1

句子示例：① 0

② 1

**＜非零数字＞ ::= １｜．．．｜９**

作用：定义了非零数字

限定条件：非零数字里没有0

句子示例：9

**＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'**

作用：定义了字符的形式

限定条件：字符必须以单引号开始，同时必须以单引号结束，同时两个单引号之间可以为加法运算符或乘法运算符或字母或数字；说明’ ‘不是字符

句子示例：① ‘+’

② ‘\*’

③ ‘\_’

④ ‘0’

**＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII字符｝"**

作用：定义了字符串的形式以及可以出现在字符串中的合法的ASCII字符

限定条件：字符串必须以双引号开始，以双引号结束，且双引号不是字符串中允许出现的ASCII字符；另外该文法中的字符串不是该文法字符的简单拼接；且合法的字符在字符串中可以出现一次或多次，因此””也是合法的字符串

句子示例：① “”

② “a”

③ “ ”

④ “’”

⑤ “\”



**＜程序＞ ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞**

**|＜无返回值函数定义＞}＜主函数＞**

作用：

限定条件：

句子示例：

**＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}**

作用：说明了常量说明的方式

限定条件：一个常量说明必须以’const’开头，’const’后为常量定义，在常量定义结束后，必须有一个’;’表示结束；常量说明可以为一个或多个

句子示例：① const int a = 1;

② const char b = ‘0’, c = ‘B’;

const char d = ‘1’;

const int e = -0;

**＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}**

**| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}**

作用：说明了常量定义的两种方式

限定条件：常量定义只有以’int’或以’char’开头两种形式，后面跟一个标识符表示常量的名字，再跟一个’=’表示赋值符号，再之后若以’int’开始则为整数，若以’char’开头则为字符；一个类型可以一次定义多个常量，两个常量定义直接用’,’分隔；但一次不能定义两个或以上的不同类型的常量，如int a = 1, char b = ‘b’是非法的

句子示例：① int a = 1

② char b = ‘0’, c = ‘B’

**＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝**

作用：说明了无符号整数的组成方法

限定条件：无符号整数必须以非零数字开始，在非零数字后面可以是合法长度内的数字，也就是说0不是无符号整数

句子示例：101

**＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０**

作用：说明了整数的组成方法

限定条件：整数的开头可以是’+’或’-‘以表示是正数还是负数，也可以没有正负号，正(或负)号之后为无符号整数或者0，因此+0，0，-0都是整数

句子示例：① 0

② +0

③ -0

**＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝**

作用：定义了标识符的合法成分

限定条件：标识符必须以字母开始，后面可以是在合法长度内的字母或数字或字母和数字的合法组合

句子示例：① \_\_\_\_\_\_\_\_\_\_

② \_12345

③ a\_\_\_\_\_\_a

**＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞**

作用：说明了声明头部的组成方式

限定条件：声明头部只能以int或char开始，后面跟一个标识符

句子示例：① int a

② char b

**＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}**

作用：说明了变量说明的形式

限定条件：变量说明由一个或多个变量定义组成，每个变量定义后面必须以’;’结束

句子示例：int a[10];

char b[10], c;

**＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ )}**

作用：说明了变量定义的组成方法

限定条件：变量定义必须以类型标识符开始，类型标识符后面是单个标识符(定义单个变量)，也可以定义一个数组，由标识符后面出现’[‘说明要定义数组，接着是一个无符号整数来说明数组的长度，最后以’]’结束；可以同时定义多个同一类型的变量，每个变量之间用’,’隔开；注意变量定义时不能赋值

句子示例：① int a[10]

② char b[10], c

**＜常量＞ ::= ＜整数＞|＜字符＞**

作用：说明了常量的合法组成

限定条件：常量只能为整数或字符

句子示例：‘0’

**＜类型标识符＞ ::= int | char**

作用：说明了合法的类型标识符

限定条件：类型标识符只能为int或char

句子示例：① int a

② char b

**＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’ //第一种选择为有参数的情况，第二种选择为无参数的情况**

作用：说明了有返回值函数定义的组成

限定条件：有返回值函数定义必须以声明头部开始，若函数有参数，则以’(‘表示开始读取参数，读完参数以后以’)’表示结束，接着以’{‘表示开始读取复合语句，复合语句结束以后以’}’表示结束

句子示例：

**＜无返回值函数定义＞ ::= void＜标识符＞(’＜参数＞‘)’‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞‘}’ //第一种选择为有参数的情况，第二种选择为无参数的情况**

作用：

限定条件：

句子示例：

**＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞**

作用：

限定条件：

句子示例：

**＜参数＞ ::= ＜参数表＞**

作用：说明了参数的由参数表组成

限定条件：无

句子示例：int a, char b

**＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}**

作用：说明了参数表的组成

限定条件：参数表必须以类型标识符开始，后面跟一个标识符，若有多个参数，则用’,’隔开

句子示例：int a, char b

**＜主函数＞ ::= void main‘(’‘)’‘{’＜复合语句＞‘}’**

作用：

限定条件：

句子示例：

**＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}**

作用：此处表述了表达式由项组成的具体方式

限定条件：包含多种情况，值得注意；第一个项前面的+、-是表示如果第一个项需要是负数或正数，在数字前加上符号；后面的+、-基本上为运算符的+、-

句子示例：① + 100 \* 50 / (1 + 1) + 2

② - 1 + 1 \* a[10]

**＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}**

作用：说明了项的组成方式

限定条件：项至少有一个因子，也可以是若干个因子之间进行乘法或除法运算的结果

句子示例：100 \* 50 / 2

**＜因子＞ ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’|‘(’＜表达式＞‘)’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞**

作用：说明了因子的组成方式

限定条件：因子可以是一个标识符；也可以是数组元素，数组元素以标识符开始，然后以’[‘表示读入表达式求数组下标，表达式结束以后以’]’表示结束；也可以是以’(‘开头表示读入表达式，表达式结束以后以’)’表示结束；也可以是一个整数；也可以是一个字符；也可以是有返回值函数调用语句

句子示例：① a

② b[\_c]

③ (d + e \* 100)

④ 100

⑤ ‘+‘

⑥

**＜语句＞ ::= ＜条件语句＞｜＜循环语句＞| ‘{’＜语句列＞‘}’｜＜有返回值函数调用语句＞; |＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜情况语句＞｜＜返回语句＞;**

作用：说明了语句的组成

限定条件：语句可以是条件语句；循环语句；或者以’{‘开头表示开始读入语句列，语句列结束以后以’}’表示结束；或者是有返回值函数调用语句并且在该语句之后有’;’表示结束；或者是无返回值函数调用语句并且在该语句之后有’;’表示结束；或者是赋值语句并且在该语句之后有’;’表示结束；或者是读语句并且在该语句之后有’;’表示结束；或者是写语句并且在该语句之后有’;’表示结束；**或者只有一个’;’**；或者为情况语句；或者为返回语句并且在该语句之后有’;’表示结束

句子示例：

**＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞**

作用：

限定条件：

句子示例：

＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞else＜语句＞

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真

＜循环语句＞ ::= while ‘(’＜条件＞‘)’＜语句＞

＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞[＜缺省＞] ‘}’

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

＜情况子语句＞ ::= case＜常量＞：＜语句＞

＜缺省＞ ::= default : ＜语句＞

＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

**＜语句列＞ ::= ｛＜语句＞｝**

作用：说明了语句列的组成

限定条件：语句列由0条或1条或多条语句组成

句子示例：

**＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’**

作用：说明了读语句的组成

限定条件：读语句必须以scanf开始，接着以’(‘表示标识符的开始，可以是一个或多个标识符，标识符之间用’,’隔开，标识符结束以后以’)’表示结束

句子示例：scanf(a,b)

**＜写语句＞ ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’**

作用：说明了写语句的组成

限定条件：写语句必须以printf开始，然后以’(‘表示要写的内容，要写的内容可以只有字符串，也可以只有表达式，也可以两者一起写，当既有字符串又有表达式时，字符串必须在表达式之前，两者之间用逗号隔开，在需要写的内容之后以’)’表示结束

句子示例：① printf(“abcde”, 1+1)

② printf(“hello\_world”)

③ printf(a[0])

**＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]**

作用：说明了返回语句的组成

限定条件：返回语句以return开始，后面可以跟’(‘表示有表达式，表达式结束以后以’)’表示结束；也可以只有一个return

句子示例：① return

② return 1

**附加说明：**

**（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符**

**（2）标识符不区分大小写字母**

**（3）写语句中的字符串原样输出**

**（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句**

**（5）数组的下标从0开始**