本课程的实验，是根据给定的文法实现一个小型编译器。该编译器的实现分成几个阶段来完成，请根据每次作业的提示，逐步增加该编译器的模块，最终形成一个生成特定目标代码的完整编译器。

开始编写编译器代码之前，需要仔细阅读和分析给定的文法、设计编译器的总体结构和各模块之间的接口、考虑编译的每个阶段可能出现的错误类型。为此，第一项作业从文法的解读开始。

请仔细阅读下面的文法，要求对文法中每条规则所定义的语法成分进行分析，了解其作用、限定条件、组合情况和可能产生出的句子，在此基础上，编写4-6个测试程序，要求所有测试程序能共同覆盖所有语法规则及每条规则内的常见组合情况，每个测试程序有且仅有10个写语句（每个测试程序的第一条写语句请用printf '('＜字符串＞ ')'输出自己的学号）。请提供每个测试程序的输入数据(有<读语句>则提供，否则无需提供)、输出数据（若输入输出数据没有正确提供，评测时会报错），放到文件中，按下述要求为文件命名：

测试程序及对应的输入输出数据文件分别为

         testfile1.txt   testin1.txt   testout1.txt

               testfile2.txt   testin2.txt   testout2.txt

                                                ...

               testfilen.txt    testinn.txt   testoutn.txt

将以上文件打包为rar或zip文件后上传（请直接打包文件，不用置于文件夹中打包）。

     注：本学期实验的所有作业都针对此文法，后续作业需在上一次作业的基础上进行增量式开发

＜加法运算符＞ ::= +｜-         /\*测试程序需出现2种符号\*/  
＜乘法运算符＞  ::= \*｜/         /\*测试程序需出现2种符号\*/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==    /\*测试程序需出现6种符号\*/  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z   /\*测试程序需出现下划线、小写字母、大写字母3种情况\*/

＜数字＞   ::= ０｜１｜．．．｜９                        /\*测试程序至少出现1次数字\*/  
＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'   /\*测试程序至少出现4种类型的字符\*/

＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝" //字符串中要求至少有一个字符

                                                                                                      /\*测试程序至少出现1次字符串，不必覆盖所有字符\*/

＜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＞   /\*测试程序只需出现有/无常量说明、有/无变量说明、有/无函数定义的情况，不必考虑其排列组合\*/

＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}    /\*测试程序需出现一个const语句、2个或2个以上const语句2种情况\*/  
＜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＞}   /\*测试程序需出现int和char 2种类型的常量，需出现1个常量定义、2个或2个以上常量定义2种情况，不必考虑排列组合\*/

＜无符号整数＞  ::= ＜数字＞｛＜数字＞｝  /\*测试程序需出现1位整数、2位及2位以上整数2种情况\*/  
＜整数＞        ::= ［＋｜－］＜无符号整数＞  /\*测试程序需出现不带+/-号的整数、带+和-号的整数\*/

＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

               //标识符和保留字都不区分大小写，比如if和IF均为保留字，不允许出现与保留字相同的标识符

              /\*测试程序需出现只有1个字母的标识符，有后续字母或数字2种情况\*/

＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞   /\*测试程序需出现int 和 char2种类型的声明头部\*/

＜常量＞   ::=  ＜整数＞|＜字符＞    /\*测试程序需出现整数和字符2种情况的常量\*/

＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}   /\*测试程序需出现只有1项变量定义、2项及2项以上变量定义2种情况\*/

＜变量定义＞ ::= ＜变量定义无初始化＞|＜变量定义及初始化＞  /\*测试程序需出现变量定义无初始化和有初始化2种情况\*/

＜变量定义无初始化＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'|＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']'|＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']' )}

                 //变量包括简单变量、一维、二维数组，＜无符号整数＞表示数组各维元素的个数，其值需大于0，数组元素按行优先存储

                 //变量没有初始化的情况下没有初值

                 /\*测试程序需出现一维数组定义、二维数组定义2种情况；需出现一个类型标识符后有1项、2项及2项以上变量2种情况，不必考虑排列组合\*/

＜变量定义及初始化＞  ::= ＜类型标识符＞＜标识符＞=＜常量＞|＜类型标识符＞＜标识符＞'['＜无符号整数＞']'='{'＜常量＞{,＜常量＞}'}'|＜类型标识符＞＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']'='{''{'＜常量＞{,＜常量＞}'}'{, '{'＜常量＞{,＜常量＞}'}'}'}'

                 //简单变量、一维、二维数组均可在声明的时候赋初值，＜无符号整数＞表示数组各维元素的个数，其值需大于0，数组元素按行优先存储，＜常量＞的类型应与＜类型标识符＞完全一致，否则报错；每维初值的个数与该维元素个数一致，否则报错，无缺省值；

                 /\*测试程序需出现一维数组定义、二维数组定义2种情况；需出现一个类型标识符后有1项、2项及2项以上变量2种情况，不必考虑排列组合\*/

＜类型标识符＞      ::=  int | char  /\*测试程序需出现int、char 2种类型标识符\*/

＜有返回值函数定义＞  ::=  ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'       /\*测试程序需出现有返回值的函数定义\*/  
＜无返回值函数定义＞  ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'      /\*测试程序需出现无返回值的函数定义\*/  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞                /\*测试程序的复合语句需出现有和无常量说明2种情况、有和无变量说明2种情况，不必考虑其排列组合\*/

＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞   /\*测试程序需出现无参数、1个参数、2个及2个以上参数3种情况\*/  
＜主函数＞    ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’                                                          /\*每个测试程序有且仅有1个主函数\*/

＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}   //[+|-]只作用于第一个<项>

                                               /\*测试程序的表达式需出现项之前无符号、有+和-号的情况；表达式只有1个项、2个及2个以上项2种情况，不必考虑排列组合\*/  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}     /\*测试程序的项需出现只有1个因子、2个及2个以上因子2种情况  
＜因子＞    ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|＜标识符＞'['＜表达式＞']''['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞

                //char 类型的变量或常量，用字符的ASCII 码对应的整数参加运算

                //＜标识符＞'['＜表达式＞']'和＜标识符＞'['＜表达式＞']''['＜表达式＞']'中的＜表达式＞只能是整型，下标从0开始

                //单个＜标识符＞不包括数组名，即数组不能整体参加运算，数组元素可以参加运算

               /\*测试程序的因子需出现7种情况\*/  
＜语句＞    ::= ＜循环语句＞｜＜条件语句＞| ＜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＞;｜＜情况语句＞｜＜空＞;|＜返回语句＞; | '{'＜语句列＞'}'          /\*测试程序需出现11种语句\*/

＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞|＜标识符＞'['＜表达式＞']''['＜表达式＞']' =＜表达式＞

                 //＜标识符＞＝＜表达式＞中的<标识符>不能为常量名和数组名

                /\*测试程序需出现给简单变量赋值、一维数组元素赋值、二维数组元素赋值3种情况\*/  
＜条件语句＞  ::= if '('＜条件＞')'＜语句＞［else＜语句＞］   /\*测试程序需出现有else和无else 2种形式的条件语句\*/  
＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞

                  //表达式需均为整数类型才能进行比较

                 /\*测试程序中需出现条件\*/  
＜循环语句＞   ::=  while '('＜条件＞')'＜语句＞| for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞')'＜语句＞

                 //for语句先进行条件判断，符合条件再进入循环体

               /\*测试程序中需出现while和for 2种循环语句，for语句应出现+步长和-步长2种情况\*/  
＜步长＞::= ＜无符号整数＞

＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞＜缺省＞‘}’     /\*测试程序需出现情况语句\*/

＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}                       /\*测试程序需出现只有1个情况子语句、2个及2个以上情况子语句2种情况\*/

＜情况子语句＞  ::=  case＜常量＞：＜语句＞                            /\*测试程序中需出现情况子语句\*/

＜缺省＞   ::=  default : ＜语句＞                                                /\*测试程序中需出现缺省语句\*/

//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  
＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'         /\*测试程序需出现有返回值的函数调用语句\*/  
＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'         /\*测试程序需出现无返回值的函数调用语句\*/

//函数调用时，只能调用在之前已经定义过的函数，对是否有返回值的函数都是如此  
＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞

                //实参的表达式不能是数组名，可以是数组元素

                //实参的计算顺序,要求生成的目标码运行结果与Clang8.0.0 编译器运行的结果一致

               /\*测试程序中需出现无实参、1个实参、2个及2个以上实参3种情况\*/  
＜语句列＞   ::= ｛＜语句＞｝ /\*测试程序的语句列需出现无语句、有语句2种情况\*/  
＜读语句＞    ::=  scanf '('＜标识符＞')'

               //从标准输入获取<标识符>的值，该标识符不能是常量名和数组名  
               //生成的PCODE或MIPS汇编在运行时，对每一个scanf语句，无论标识符的类型是char还是int，末尾均需回车；在testin.txt文件中的输入数据也是每项在一行

              //生成MIPS汇编时按照syscall指令的用法使用

             /\*测试程序中需出现读语句\*/  
＜写语句＞    ::= printf '(' ＜字符串＞,＜表达式＞ ')'| printf '('＜字符串＞ ')'| printf '('＜表达式＞')'

               //printf '(' ＜字符串＞,＜表达式＞ ')'输出时，先输出字符串的内容，再输出表达式的值，两者之间无空格

              //表达式为字符型时，输出字符；为整型时输出整数

              //＜字符串＞原样输出（不存在转义）

              //每个printf语句的内容输出到一行，按结尾有换行符\n处理

             /\*测试程序中需出现3种形式的写语句\*/  
＜返回语句＞   ::=  return['('＜表达式＞')']

              //无返回值的函数中可以没有return语句，也可以有形如return;的语句

             //有返回值的函数只要出现一条带返回值的return语句（表达式带小括号）即可，不用检查每个分支是否有带返回值的return语句

             /\*测试程序中需出现有返回语句和无返回语句2种情况，有返回语句时，需出现有表达式和无表达式2种情况\*/

另：关于类型和类型转换的约定：

1. 表达式类型为char型有以下三种情况：

         1）表达式由<标识符>、＜标识符＞'['＜表达式＞']和＜标识符＞'['＜表达式＞']''['＜表达式＞']'构成，且<标识符>的类型为char，即char类型的常量和变量、char类型的一维、二维数组元素。

         2）表达式仅由一个<字符>构成，即字符字面量。

         3）表达式仅由一个有返回值的函数调用构成，且该被调用的函数返回值为char型

  除此之外的所有情况，<表达式>的类型都是int

2. 只在表达式计算中有类型转换，字符型一旦参与运算则转换成整型，包括小括号括起来的字符型，也算参与了运算，例如(‘c’)的结果是整型。  
3. 其他情况，例如赋值、函数传参、if/while条件语句中关系比较要求类型完全匹配，并且＜条件＞中的关系比较只能是整型之间比，不能是字符型。