CompilerNKCS

实现一个简单c语言编译器

目标：编译器演示程序，将C语言程序编译为目标代码————汇编程序，用汇编器转换成二进制程序后运行无误

基本要求：

```
数据类型：int

语句：注释，声明，赋值，循环（while 和 for），判断（if），输入输出

算术运算：+，-，*，/，%，^

关系运算：==，>，<，>=，<=，!=

逻辑运算：&&（与），||（或），!（非）
```

完成功能：

```
词法分析、语法分析、类型检查、代码优化、错误分析、汇编程序
```

加分项：

```
支持数组运算 一维数组

支持指针运算 一维指针
```

操作方法：

如果您想一次性跑完下面的命令，在根目录下运行`./pao.sh`能快速执行完下面的命令

命令解析

```
make grammar    编译lexxa.l、grammar.y

make parser     生成可执行文件

make build      生成build文件，一键拷贝文件，供后续中间代码生成

cd output       进入输出文件夹，可以看到名为parser的可执行文件，它包括了语法分析和词法分析以及中间代码生成
```

```
./parser test/xx.c    运行`output/test`文件夹下面的词法分析和语法分析

make clean       删除生成的文件，注意要在根目录下执行，而不是在output文件夹下
```

程序解析

文件阅读顺序：

词法分析： lexxal.l
语法分析： grammar.y
中间代码生成： common/util/InterMediate.h(cpp)
四元式生成： common/util/Quad.h(cpp)
汇编代码生成： common/util/AsmGenerator.h(cpp)
其中在grammar.y文件内会调用lexxal进行词法分析，因此词法分析和语法分析的所有的操作都在grammar.y文件内被定义

语法分析所用到的c文件

```
#include "./tables/symbol.h"
#include "./trees/ASTNode.h"
#include "./trees/StmtASTNode.h"
#include "./trees/LiteralASTNode.h"
#include "./trees/OpASTNode.h"
#include "./trees/VarASTNode.h"
#include "./trees/DefVarASTNode.h"
#include "./trees/LoopASTNode.h"
#include "./trees/ConditionalASTNode.h"
```

上述的c文件看似很多，但是只是为了条件清晰，将每个语法树的类型单独创建一个文件体，
其中根文件为 *./trees/ASTNode.h(cpp)* ，它定义了抽象类型，其它的文件均继承于它，所以可以从这个文件看起

. . . . . .

symbol.h和symbol.cpp中定义了词法分析生成符号表的主要函数
在grammar.y文件中会调用词法分析器进行词法分析，然后在grammar.y文件进行语法分析,全部的程序逻辑在`grammar.y`的main函数中实现