关于qinling cpu部分，避免遗忘。

设计上主要考虑能否进行烧录，以及bin文件的读写功能，原来的编译器只支持etb文件的生成，此文件类似于hex文件

设计实现代码如下：

QByteArray ExecutableImage::toEtbFileData(){

int length = this->length();

reorder();

QByteArray bin;

QBuffer buffer(&bin);

buffer.open(QBuffer::ReadWrite);

QDataStream stream(&buffer);

QString titleText = "OpenQinling-CPU ETB file\r\n";

titleText += "title:\r\n"+this->toString();

titleText += "bin:\r\n";

QByteArray arr = titleText.toUtf8();

for(int i = 0;i<arr.length();i++){

stream <<(int8\_t)arr[i];

}

stream <<(int8\_t)0;

stream << length;

stream << (1-length);//校验码

for(int i = 0;i<length;i++){

stream << length-i;//校验码

stream << this->at(i).baseAdd;

stream << this->at(i).len;

stream << this->at(i).isZiData;

stream << this->at(i).data;

}

buffer.close();

return bin;

}

关键内容在于后半部分的for循环，用于识别block，将每个block的内容放置于stream中，最后返回到bin中。

但是在soc设计中无法识别此类文件，需要bin文件生成。

故本人基于此开发了转换bin文件的函数。

目的在于用于将代码放置到存储器中。用于总线的读写。

所以需要修改编译器的内容，故需要先了解整个编译器的架构，以及明白我们需要修改的部分。

编译器一般由：

1. 词法分析器：
2. 语法分析器：
3. 语义分析器：
4. 中间代码生成器
5. 优化器：
6. 目标代码生成器：
7. 汇编器：
8. 链接器：

等组成，具体实现中不必拘泥于某个部分需要怎么设计。在进行设计时，不同的编译器可以自由组合或拆解其中的步骤。

比如在Qinling的编译器设计中，语法分析和中间代码生成器是一起完成的。

好的，回到正题，我们还是一步步开始讨论代码是如何设计的吧。

此编译器的设计实现：

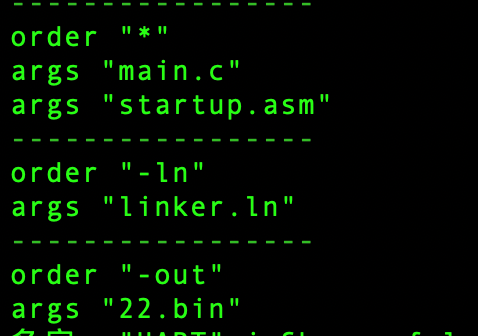
1. 首先，编译器调用了命令行解释函数对输入的终端指令进行分析

命令终端输入为：



023A\_CC.exe main.c start\_up.asm -ln linker.ln -out 22.bin

命令行解析order 和args



然后终端解析函数会解析每一个出现的order。



例如此代码，识别是否存在-work这个order

然后识别完了后会进入其他的处理的函数：

因为不只是可以生成bin文件，如果想生成汇编代码，中间语言文件等，可以使用命令例如：

-asm main.c -out main.asm



这段代码首先识别命令给的是否正确，比如-asm后是否是有一个.c，以及检测是否

-asm前面无其他order。

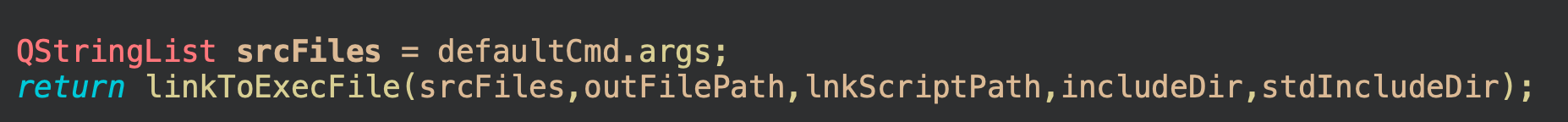
最后调用compileASM();将main.c转换为main.asm

==============

然后如果都是不是。就进入正常的转换代码，

return linkToExecFile(srcFiles,outFilePath,lnkScriptPath,includeDir,stdIncludeDir);

传入的参数包括：



希望未来的我可以懂！！，这里的defaultCmd.args就是因为最开始的order有一个\*号，他后面的各种.c .asm都是属于这个order的args,所以这里传入参数srcFiles也就是里面都是输入的源文件的名字。说的还不错，挺清楚了。

然后就是输出文件🪖路径，一般是指定了，例如22.bin，虽然说的是路径，但是不是绝对路径，一般就是输出文件名字。

最后就是链接lnkScriptPath,也就是链接的文件的名字了，

includeDir可以设置头文件的搜索路径，

stdIncludeDir也是头文件搜索路径，目前着两个我都没有指定过，都放到一个文件夹里面就行。

然后就是跳转到了正式执行函数，将源文件转换为bin文件。

这个函数会解析传入的文件，包括源文件，链接文件，并检查输出文件的后缀是否正确，需要为etb或者bin。

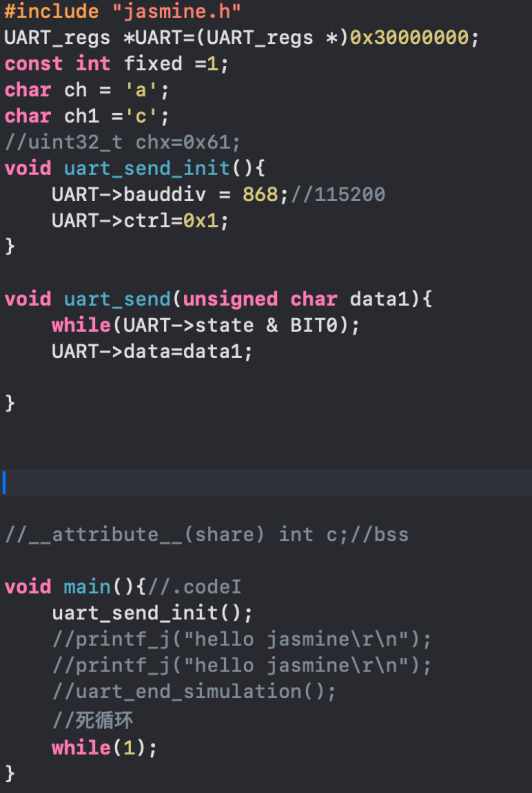
链接文件的解析是最后一步的事情，在此先介绍源文件的代码解析，大的步骤包括，中间语言生成，汇编生成，最后是block的生成，然后是链接。

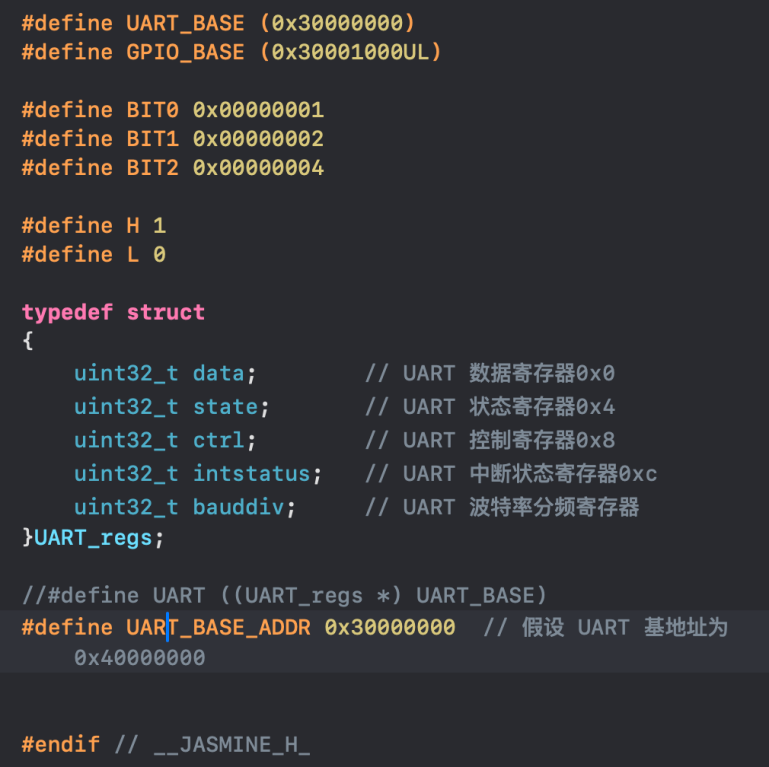
由于写到这里，我其实并不完全清楚这个编译器的前端的细节部分，所以我写的过程顺便来阅读代码。希望未来的我看的懂。

首先是使用for循环去解析源文件。

例如我们接下来对于一个简单的main.c做分析。

这个mian.c的代码：





这个工程的内容还是比较丰富的，应该可以说明整个过程。

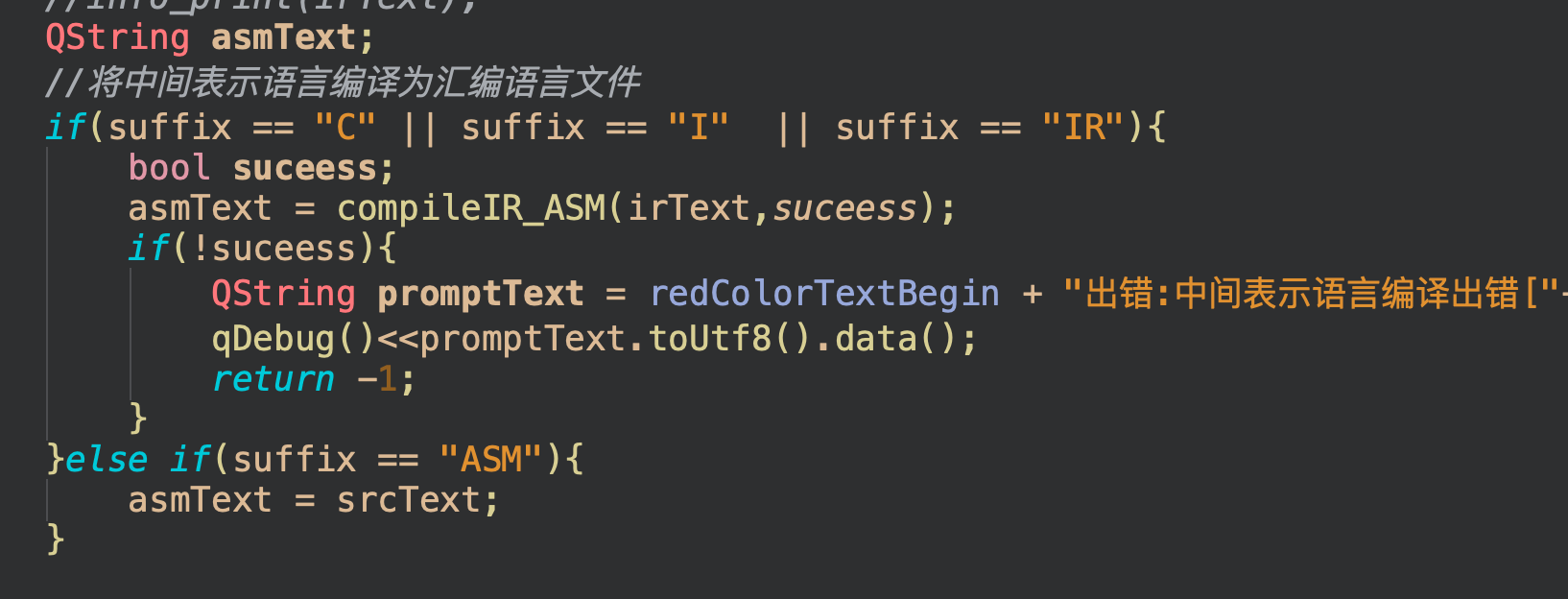
首先会提取源文件的内容：



然后是中间语言生成：



最后放到了irText



然后用汇编生成



最后，用汇编文件生成静态库文件。

说完总体的逻辑后，我们来详细讨论每一个步骤，首先是生成中间语言的过程，这个过程一般都是先进行词法分析。

我们进入函数调用查看。



这是函数中的文件，我们首先分析一下函数的参数的调用

最重要的是src源码，

然后是srcPath，就是本目录

compileType:不知道有什么用

includeDir就是头文件的搜索目录，在前面有提到，可以指定，也可以默认为本目录。

stdIncludeDir:也可以指定

exterdAttributes:无用

后续是一些结果返回，和可选项，目前只有32位和内存大端存放的方式。

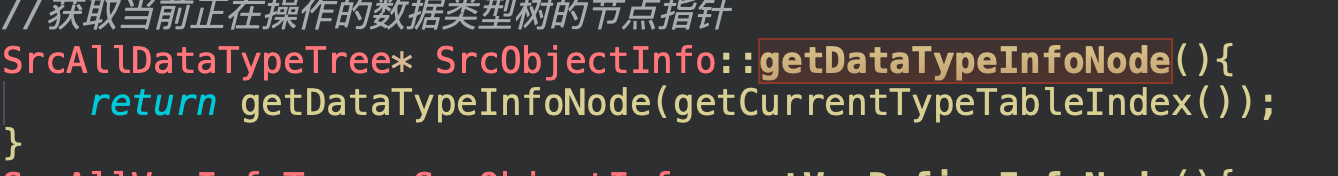
需要介绍一个重要的类，这个数据结构包含了一个源文件的所有信息，所以是非常重要，后续关于这个源文件的操作估计都和这个类有关系，需要了解其中的内容。

//c源文件信息

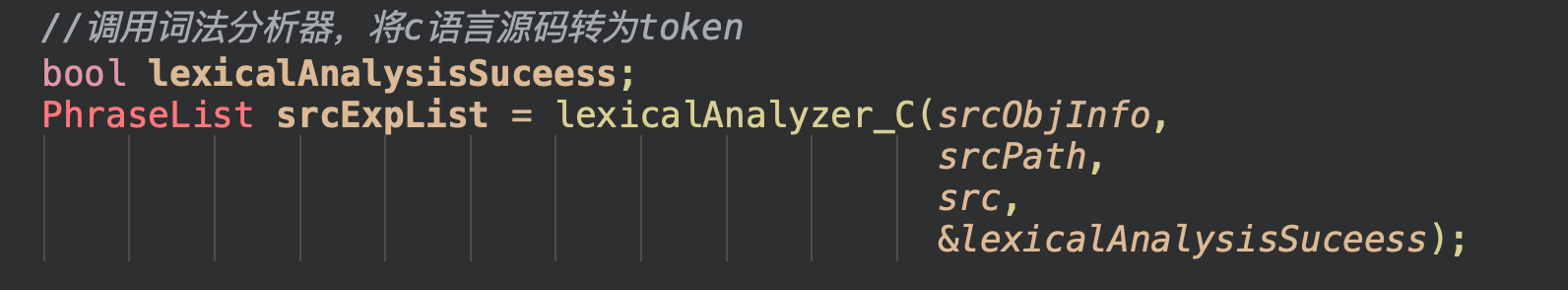
struct SrcObjectInfo{



进入函数后，首先把c语言中的类型的定义全部注册到一个类型表中，这里原本有点不清楚，我需要详细解释一下，这里的执行过程首先是调用了函数getDataTypeInfoNode()



这个函数首先调用了，然后返回一个类指针，然后这个类中有个成员是thisNodeTable，进入registerCbaseDataType函数后，用别名的方式给这个成员赋值。



然后嗲用词法分析，会返回一个词组列表。

PhraseList是一个词组列表，类容器，里面将每一个词组都存放到一个容器成员中。