# 编译器设计与实现

## 实验目标

深入理解编译器的典型框架结构及各个构件之间的衔接关系，加深对课堂讲授的概念、基本算法和数据结构等理论知识的理解与认识。通过手动创建并实现一个编译器，提高对程序设计语言，机器语言的认知，加强学生计算机系统能力的培养。

## 实验要求

（1） 从Github下载并运行课程提供的编译器框架。框架下载地址：

https://github.com/jiweixing/bit-minic-compiler

有关该框架的详细介绍见附录A。

（2）选择一个C语言的语法子集，使用Java语言或者C语言编写并替换框架已有模块。

（3）其中词法分析、语法分析和中间代码生成为必选模块，代码优化、X86或者MIPS汇编代码生成、汇编及二进制代码模拟运行为可选模块。要求支持C语言出现的所有单词类，所选语法子集包括一种分支语句和循环相关的文法。例如输入程序里面可以有if语句和for循环语句。

（4）根据实验过程和实验结果撰写实验报告，包括每个模块主要的数据结构和分析方法，模块的输入和输出结果。

## 提交内容

包含框架的C语言迷你编译器源代码和运行示例。提交目录包括src、bin和doc三个子目录。其中src为源代码目录，bin为可执行程序目录，doc为文档目录。请每个同学按照“班号\_学号\_姓名”的命名方式将整个目录打包，班级统一提交给老师。

## 附录A：bit-minic-compiler的框架结构

### 基本概况

bit-minic-compiler是使用Java语言实现的一个C语言编译器的框架。该框架的主要特点在于：

（1）该框架将C语言的编译过程划分为多个阶段，两个不同的阶段通过XML文件进行交互。这样设计的目的是使学生能够更好的了解每个阶段的工作原理，输入数据和输出数据。通过观察模块之间的交互和衔接，能够更直观的了解编译器的工作过程。

（2）该框架包含了编译器各个阶段的内部实现，学生可以直接运行该框架，查看多个阶段之间的输入和输出。内部集成的各个模块的源代码是不可见的，仅供学生自己实现各个模块时参考。

（3）在使用框架的过程中，学生可以自由选择使用内部集成的模块或者自己设计的模块。其原因及好处在于，编译器各个阶段是互相依赖的，如果前面部分实现不好，后续工作较难进行，但是基于该框架，进行后端的实验时，可以直接选择使用内部集成的前段模块，从而节省了时间。

（4）该框架集成了MIPS的汇编器和模拟器MARS，生成代码后可以直接调用该模块对生成的代码进行验证。如果验证成功，则可以与体系结构和组成相关课程实验进行衔接，将生成的代码在自己设计的目标系统上运行。

C源程序

预处理程序

C源程序

词法分析器

语法分析器

语义分析与代码生成

代码优化

汇编

代码生成

汇编与模拟运行

属性字符流

语法树

四元式

四元式

汇编代码

二进制代码

如下表给出了各个阶段输入和输出文件的定义和命名规则：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 输出 | 功能描述 |
| 预处理 | xxx.c | xxx.pp.c | 删除无用注释和空格，宏替换与文件包含 |
| 词法分析 | xxx.pp.c | xxx.token.xml | 词法分析，生成属性字符流 |
| 语法分析 | xxx.token.xml | xxx.tree.xml | 语法分析，生成语法树 |
| 语义分析 | xxx.tree.xml | xxx.tree2.xml | 语义检查 |
| 中间代码生成 | xxx.tree2.xml | xxx.ic.xml | 生成四元式列表 |
| 代码优化 | xxx.ic.xml | xxx.ic2.xml | 实施常量合并等代码优化 |
| 目标代码生成 | xxx.ic2.xml | xxx.code.s | 生成x86或者MIPS汇编代码 |

需要说明的是，如果生成的是x86的汇编代码，则可以使用x86汇编器将汇编代码翻译为机器代码之后执行，如果生成的是MIPS汇编代码，则可以使用框架集成的MARS对其进行汇编和模拟执行。

该框架的实现为学生提供了一个非常好的参考，同时节省了学生大量的时间，使其能够更加集中的进行课程相关的实验。

### 运行环境

由于框架基于Java语言开发，所以框架的运行需要Java运行时环境。目前项目中配置为JRE 1.7，但是也可修改为更低或者更高的版本。

### 框架调用接口

考虑到学生对各种程序设计语言的掌握程度不同，框架提供了两种模式对不同的目前的框架进行扩展。由于框架本身采用Java语言设计，因此采用Java语言重新实现各个模块是最直接和有效的方法。框架为编译每个阶段对应的模块定义了接口，内部集成的模块实现通过实现该接口完成特定的功能。同理，用户程序也可以实现同样的接口，已完成接口一致性。如下对接口的定义一一进行说明：

（1）预处理接口定义

public interface MiniCCPreProcessorInter {

public void run(String iFile, String oFile);

}

（2）词法分析接口定义

public interface MiniCCScannerInter {

public void run(String iFile, String oFile);

}

（3）语法分析接口定义

public interface MiniCCParserInter {

public void run(String iFile, String oFile);

}

（4）语义分析接口定义

public interface MiniCCSemanticInter {

public void run(String iFile, String oFile);

}

（5）中间代码生成接口定义

public interface MiniCCICGenInter {

public void run(String iFile, String oFile);

}

（6）代码优化接口定义

public interface MiniCCOptInter {

public void run(String iFile, String oFile);

}

（7）目标代码生成接口定义

public interface MiniCCCodeGenInter {

public void run(String iFile, String oFile);

}

用户只需要定义一个新的类，实现相应的接口，并在MiniCCompiler中创建自己的类即可完成自己实现的功能模块的调用。

在bit-minic-compiler框架下，用户选择使用其他的语言实现相应的模块也是可以的，例如C/C++或者Python。框架中定义并实现了与MiniCCompiler功能等价的一个类MiniCStubCompiler，该方法通过调用第三方应用完成整个编译流程的控制。如果用户用C/C++语言实现了相应的模块，只要把程序编译为相应的可执行程序，并放入到指定的目录下，则框架执行过程中会调用该可执行程序，将输入文件和输出文件以命令行参数的方法传入。选择别的语言仍然使用bit-minic-compiler的原因在于，框架提供了众多通用的管理和通用方法，能够大幅减少用户的工作量，使得用户集中注意力解决最主要的问题。例如出错处理和符号表的管理等。

如有其它关于该框架的使用问题及建议，请与北京理工大学计算机学院编译课程组的老师联系：

计卫星， jwx@bit.edu.cn