摘　要

本文主要研究一个教学用的 C 语言编译器框架，旨在帮助学习《编译原理》课程的学生能够更加清晰地了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。此外，此框架也有助于学生亲自实现编译器的不同模块的功能，最终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。

本文所研究的 C 语言编译器框架中，将复杂的编译过程进行化简、分解，使之划分为不同的相对独立的模块，每个模块实现编译过程的一步或几步操作，模块之间通过一些输入输出文件进行数据关联。同时，在实现了编译器框架之后，本文设计并实现了一个内嵌编译器，即实现了编译器框架的每个模块的功能。本文研究的迷你 C 语言编译器框架由 Java 编写，跨平台性较好，能够在多个系统环境中使用。同时，框架能够支持由 Java/C/C++/Python 语言编写的具体功能模块，极大地提高了框架对其他编程语言的兼容性，使之更加容易扩展。

最后，为了测试框架的有效性和实用性，将本文所研究的 C 语言编译器框架上传到 Github 上，供正在学习《编译原理》的学生尝试使用。目前，经过一段时间的试用，得到的反馈结果较好，基本能够实现当初预计的功能，满足学生的课程学习需要。

关键字：C 语言，编译器，Java，多语言支持，编译器框架，跨平台

Abstract

This thesis is focus on a C-language compiler frame for teaching, in order to help students, who study *compiling principle*, to learn the typical fame structure and the connection of parts of a compiler more clearly, and to deepen the comprehension about the theoretical knowledge including concepts taught in the class, basic algorithms and data structure. Besides, the frame can help students to achieve the functions of different parts of a compiler, and create a compiler, so that students can improve their acknowledgement of programming language and machine language, and their abilities about the computer system.

The C-language compiler frame in this thesis simplify and divide the complicated compiling process into dependent modules. Each module achieve one or a few steps of the compiling process, and the modules are connected by some input/output files. At the same time, after finish the compiler frame, an embedded compiler is designed and achieved, which achieve the functions of each modules of the compiler frame. What's more, the Mini-C-language compiler frame is programmed by Java, whose performance is good in different platforms, and thus the frame can run in different system environment. Also, the frame supports modules programmed by Java/C/C++/Python, which highly improves the compatibility to other programming languages of the frame, and makes the frame easier to be expanded.

In the end, in order to test the validity and practicability of the frame, the C-language compiler frame is uploaded to Github, so the students learning *compiling principle* can download and try. Up to now, after trying for a period of time, the feedback is fine, which means the frame can basically achieve the functions designed at the beginning and meet the needs of students for learning.

Keyword: C-language, compiler, Java, multi-language support, compiler achievement, multi-platform

目　录

第 1 章 绪论

编译器可以看做一种“翻译器”，它的作用是将一种编程语言写的程序（源程序）翻译成为等价的另一种编程语言的程序（目标程序）。通常来讲，编译器都是将一些高级语言的程序翻译为汇编语言的程序，因此，编译器的出现与高级语言的出现密不可分。

19 世纪 50 年代，美国 IBM 公司的 John Backus 带领研究小组针对汇编语言的缺点着手研发 FORTRAN 语言，即世界上第一门高级程序设计语言。在研发这门语言的同时，针对该语言的编译器也在进行同步开发。但由于当时编译理论发展并不充分，开发工作既复杂又艰苦。与此同时，Noam Chomsky 开始研究自然语言的结构。正是通过 Chomsky 对自然语言结构的研究，才使得编译器的结构大大简化。Chomsky 通过对语言描述问题的探讨，最终提出了一种用于描述语言的系统，即 Chomsky 分类。

随着编译理论与相关技术的不断完善，人们就开始研究编译器的自动构造，这些程序被称为编译器的编译器（Compiler-compiler）这些程序中最著名的是 Yacc（Yet Another Compiler-compiler），它是由贝尔实验室为 Unix 系统编写的，是一个 LALR(1)分析器自动生成器。类似的，有限状态自动机的研究也发展了一种称为扫描程序生成器（Scanner Generator）的工具，其中最著名的是 Lex，1972 年贝尔实验室在 UNIX 上首先实现，是 UNIX 的标准应用程序。

90 年代之后，编译器逐渐发展成为 IDE 的一部分，它包括了编辑器、连接程序、调试程序以及项目管理程序等内容。然而，尽管近年来在编译原理领域进行了大量的研究，但是基本的编译器设计原理在近 20 年中都没有多大的改变，它现在正迅速地成为计算机科学课程中的中心环节。

1.1 背景介绍

《编译原理》的是计算机专业本科学生的一门必修课，也是学生必须要掌握的一门专业课。对于计算机专业的学生来讲，学习编译原理无疑是一种可以更加深入地了解计算机程序较为低层的实现机制的一个机会，通过学习编译原理，可以更加深刻地理解一个高级语言编写的程序如何被翻译成为汇编语言，以便能够在机器上执行。通过对编译原理的学习，学生可以对高级语言程序的

低层实现有更加深入的研究，能够将汇编语言与平时惯用的高级语言相互联系起来，从而对计算机体系结构有更加宏观、全面的把握。

然而，在日常的教学过程中，如何快速有效地掌握编译原理的基本理论和方法成为了学生们的一大困难，很多学生都反映编译原理课程晦涩难懂，到头来学了很多的理论却无法融会贯通，导致对编译原理的理解不够深入，无法将其理论和方法转化为一个可以运行的编译器。种种问题究其原因，就在于编译过程的复杂性太高。对于一个初学者来讲，往往很难满足编译器全过程的实现。实际上，很多学生在最终课程结束的时候，也仅仅是做了编译过程的前 2 到 3 步工作，至于后面的工作则没有多余的时间和精力去处理。这就导致了一个问题，那就是每一届的学生都只对编译过程的前几步有一定的了解，对之后一些更加重要但难度更大的步骤却毫无处理经验，最终导致无法将学习到的理论和方法进行实践，造成学习效率的下降和时间的浪费。

为了解决上述问题，本文设计与实现了一个迷你 C 语言编译器框架。这个框架将编译器的全部实现过程进行了详细划分，并对每一个划分模块都实现了一个内嵌子程序。每个模块不仅可以使用内嵌的 Java 子程序进行编译过程，也可以调用 Python/C/C++等语言编写的模块处理子程序，极大地提高了框架的灵活性和兼容性。

1.2 编译器常用结构

编译程序是十分复杂、庞大的软件，它处理的源语言程序有成千上百中，包括通用的计算机语言到各个领域的不同的专用语言，它的处理结果（目标程序）的形式既可以是另一种计算机语言，又可以是某种机器语言。因此，不同的源语言或不同的结果要求使用不同的编译器来处理。而且，现在比较主流的一些编译器，已经不仅仅是一个语言翻译程序，更是一个包含编辑器、连接程序、调试程序以及项目管理程序等功能的集成开发环境（IDE）。然而，尽管编译程序的处理过程十分复杂，不同的编译程序实现方法也各不相同，但任何编译程序的基本功能都是类似的，其基本逻辑功能以及必要的模块大致相同，即大都要经过预处理、词法分析、语法分析、语义分析和中间代码生成、代码优化、目标代码生成这些步骤，并在这些步骤中贯穿着表格管理和出错处理的功能。图 1- 1 给出了一般编译程序的总体结构。

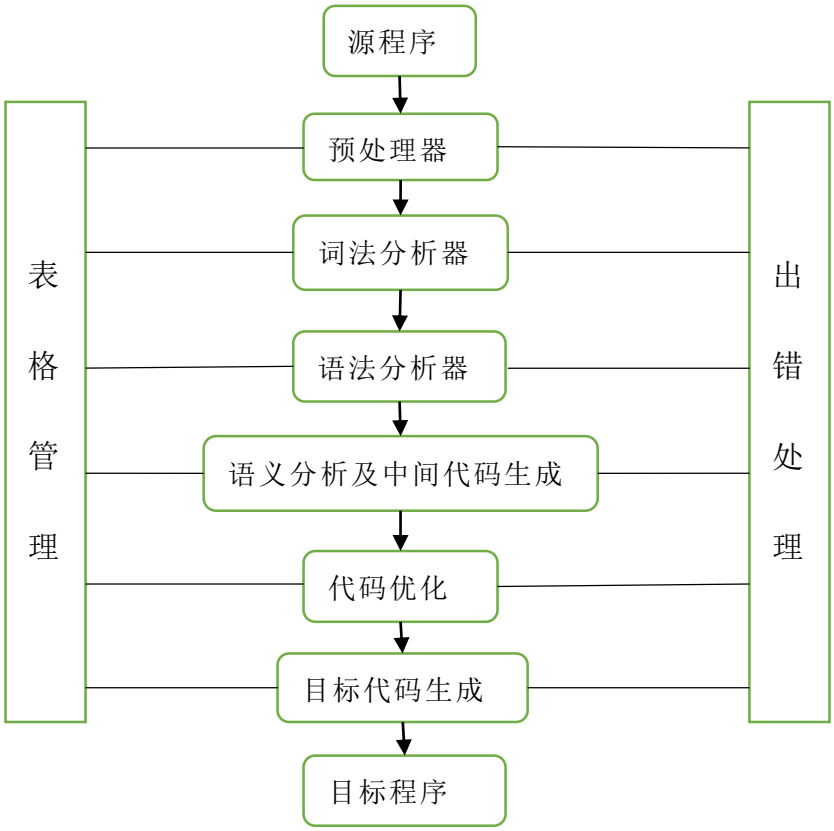

图 1- 1 编译程序总体结构

如图 1- 1 所示，一般编译器的编译过程分为预处理、词法分析、语法分析、语义分析及中间代码生成、代码优化、目标代码生成几个阶段。其中，表格管理和出错管理两个模块可以在编译的任何阶段被调用，以便辅助完成编译功能。

1.3 编译器实现技术

1.3.1 预处理

预处理程序的出现是为了减轻词法分析部分的工作量。通常情况下，词法分析接受的输入时经过预处理的源程序串，这是因为在源程序中，往往由于编程人员的习惯，会产生大量的空格、回车换行符以及注释等，这些内容是为了增加程序的可读性和方便编程，可是对程序本身确实无用的。此外，对于 C 语言来讲，程序中经常会出现宏定义、文件包含、条件编译等特性，这使得词法

分析的过程更加复杂。为了消除一些无用代码并减轻词法分析程序的处理负担，预处理程序应用而生，它一般会完成以下一些功能：

- 过滤掉程序中的注释和一些无用字符；
- 对程序进行宏替换；
- 实现文件包含的嵌入和条件编译的嵌入。

1.3.2 词法分析

词法分析是编译过程的第一个有实际意义的工作，它的任务是对输入的源程序字符串进行顺序扫描，同时根据词法规则识别出具有独立意义的单词，并生成属性字符流作为输出。

单词是指语言中具有独立意义的最小语法单位。通常情况下，对于一种程序设计语言来讲，单词可以分为以下几类：

(1) 关键字（keyword）

关键字一般是程序设计语言自身定义的，保留、不可让程序员使用的一些单词。如 C 语言中，关键字就有 int, float, double, char, unsigned, for, if, return 等。

(2) 常数(const)

常数就是指各种类型的有确定和表示的量，可以分为整型常数、浮点型常数、布尔常数、字符及字符串常数等。如“25”就表示一个整型常数，“hello world”就是一个字符串常数。

(3) 标识符(identifier)

标识符是用来表示程序中各类名字的标识，是由程序员自己创造的。如变量名、数组名、结构名、函数名等。

(4) 运算符(operator)

运算符是表示算术运算、逻辑运算、位运算等运算的字符或字符串。如 C 语言中，有+、-、*、/、=、<、<=、&、|、?:等运算符。

(5) 界限符(separator)

界限符是指程序中将上述单词分隔开的一些字符。在 C 语言中，有逗号、分号、括号、引号等。

属性字是语法分析程序对源程序中各单词处理后的输出形式，是单词在编译过程中的一种内部表示形式。通常情况下，属性字设计为二元组的形式：（单词属性，单词值）。其中，单词属性表示单词的类别，如一个单词是关键字还是标识符；单词值是编译器为单词设定的内部表示，可以缺省，与单词类别的划分有关，如对于加号“+”，如果它的单词属性是界限符，那么单词值就是“+”，而如果它的单词属性就是加号“+”，单词值就可以缺省。

词法分析器的设计于实现有很多种方法，较为常用的一种方法是利用状态转换图。对于以正则文法作为词法规则的程序设计语言，状态转换图完全可以识别出其单词。事实上，按照以下步骤就可以构造出识别单词的状态转换图：

(1)对程序语言的单词按类别构造出相应的状态转换图

(2)对各类状态转换图进行合并，形成一个能识别语言所有单词的状态转换图。

在有了状态转换图之后，接下来的工作就是进行状态转换图的实现。实现状态转换图一般有两种方法，一种是程序中心法，即把状态转换图看作一个流程图，从初态开始，为每一个状态节点编一段程序，具体程序实现可以使用条件判断语句如 switch；另一种方法是数据中心法，即把状态转换图看作一种数据结构（如状态矩阵表），编写控制程序，控制输入字符在状态转换图上运行，从而完成单词的识别。

对于词法分析器，除了自己手动编写之外，目前也有一些词法分析器的自动生成工具，其中最著名的非 Lex 莫属，它是 1972 年贝尔实验室在 UNIX 上实现的，是 UNIX 标准应用程序。Lex 编译器接收的是 Lex 源程序，它主要是对产生的词法分析器的说明和描述。Lex 编译器通过处理 Lex 源程序，可以输出一个名为 lex.yy.c 的 C 语言程序，经过 C 编译器的编译可以生成 a.out 文件，这就是一个可以运行的词法分析器。图 1- 2 表示了使用 Lex 的步骤。

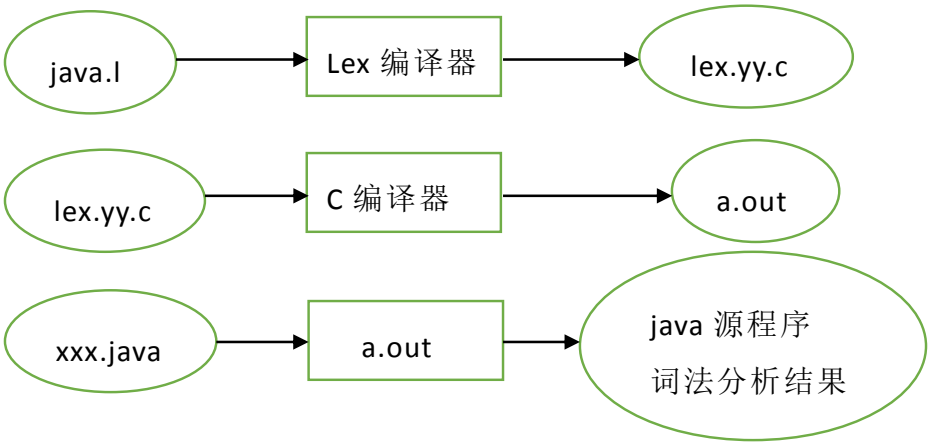

图 1- 2 Lex 生成 Java 语言编译器的过程

1.3.3 语法分析

语法分析是编译过程的核心部分，它是按照语法规则，对词法分析输出的属性字符流进行语法检查，识别出相应的语法成分，常用语法分析树的形式来表示。

语法分析的方法很多，通常情况下能够分为两大类，即自上而下分析和自下而上分析。

自上而下分析是一个推导过程，其原理是对输入串进行顺序扫描，从文法的开始符号出发，通过反复使用产生式对句型中的非终结符进行替换，逐步使推导结果与最终的输入串匹配。比较常用的两种自上而下分析方法是递归下降分析法和 LL(1)分析法。

(1)递归下降分析法是对文法的每个非终结符，根据其产生式的候选式，编写一个对应的子程序，从而完成非终结符对应语法成分的识别。状态转换图可以用来作为递归下降分析器的工具，即对文法的每个产生式分别构造状态转换图，然后针对每个状态转换图，分别编写一段子程序，进而完成整个文法的语法成分的识别。

(2)LL(1)分析法是指按自左至右的顺序扫描输入字符串，并在分析过程中产生句子的最左推导，并且在每一步推导过程中，最多只要向前查看一个输入字符，即可确定当前推导使用的文法规则。为了实现 LL(1)分析器，需要一张

分析表，一个分析栈和一个总控程序。分析表的最左侧一列表示文法的非终结符，最上面一行表示文法的终结符，中间的表项填充文法的产生式规则。分析栈中存放分析过程的文法符号。总控程序根据分析表和分析栈对输入字符串进行扫描，即读分析栈中的文法符号和输入字符串的第一个字符，通过查分析表，来判断使用哪一个产生式或进行出错处理等，从而完成对输入符号串的语法分析过程。

自下而上分析是一个归约过程，其原理是从输入串开始，反复查找当前句型中存在的文法中某个产生式的候选式，逐步进行归约，直到归约成为文法的开始符号位置。一种基本的自下而上分析思想“移进-归约”分析，该方法需要设置一个寄存文法符号的符号栈，在分析过程中将输入符号串的字符一个个地移入栈内，当栈顶符号串形成某个产生式的一个候选式时就进行归约，即将这部分符号串替换为产生式左部的符号，如果栈顶符号串没有形成候选式，那么就从输入串继续移入字符，以此类推直到整个输入串处理完毕。具体的自下而上分析方法很多，比较常用的有算符优先分析法、LR 分析法。

(1)算符优先分析法特别适用于各类表达式以及符号式语言中具有优先级特点的符号串的分析。这种方法的原理是定义文法的终结符之间的某种优先关系和结合性，通过这种关系来确定可归约串并进行归约。实际操作中，需要一个优先关系表或优先函数表，总控程序通过查表，确定符号栈中符号的优先关系，从而进行归约或出错处理。

(2)LR 分析法是指一类从左到右对输入串进行扫描的自下而上分析的方法。LR 分析法需要一个分析栈，一个 LR 分析表和一个总控程序。分析栈中包含了两部分内容：状态符号和文法符号，状态符号记录整个分析历程并预测接下来扫描的符号串，文法符号则表示在分析过程中移进或归约的符号。LR 分析表分为动作表和状态转换表，动作表记录栈顶当前状态和当前输入符号对应的分析动作，包含移进、归约、接受和出错，状态转换表则记录栈顶当前状态和当前文法符号对应的转移状态。总控程序则负责对输入符号串进行扫描，并通过查表来确定接下来的分析动作。

1.3.4 语义分析及中间代码生成

语义分析主要是对语法分析器识别的语法范畴（语法树）进行语义处理，翻译成为相应的中间代码。目前大多数编译程序实现语义分析普遍采用的一种方法是语法制导翻译方法，这种方法在语义分析过程中同时进行语义处理，其实现思想是把语言结构的属性赋给代表语言结构的文法的非终结符，属性值由附加到文法产生式规则的语义规则计算，而语义规则的计算则可以产生代码。其中涉及到两个概念，一个是语法制导定义，一个是翻译模式。语法制导定义是关于语言翻译的抽象规格说明，不处理具体细节，也不规定翻译的顺序。翻译模式则规定具体的实现途径和细节，即指明使用语义规则计算的顺序。总体来讲，语法制导定义和翻译模式是在语法分析树的基础上，遍历分析树，根据分析树对语义规则进行计算，通过生成代码、查填符号表和给出错误信息等操作来完成翻译动作。

在进行语义分析的过程中，需要不断收集、记录和使用源程序中一些语法符号的类型、特征和属性等相关信息，一般会建立并保持一批表格，即符号表，包括常数表、变量名表、数组名表及标号表等。符号表的登记项一般由名字标识符和信息两个数据项组成，其中名字标识符项用来存放标识符或其内码值，用来唯一标识该符号；信息项一般由多个子项组成，用来记录该符号的各种属性和特征。

在语义分析中，需要进行语法和语义检查，这是一种静态检查，一般包括类型检查、控制流检查、唯一性检查和关联名字检查。类型检查处理程序中类型不匹配的问题，如运算符作用于不相容的对象；控制流检查处理程序中控制流是否完整的问题，如 C 语言中一个 break 语句外并没有 while, for, switch 这样的语句进行包围，则需要报错；唯一性检查处理程序中必须唯一声明的内容是否多次声明的问题，如在同一个域中标识符必须唯一声明，case 语句的标号必须不同等；关联名字检查处理必须出现两次或多次的内容是否只出现一次的问题，如在 Ada 语言中，循环开头有名字出现，则结尾也必须再出现一次相同的名字。

中间代码是一种面向语法的、易于翻译成目标代码的源程序的等效内部表示代码。中间代码不仅是作为最终生成目标代码的过渡，它本身与具体目标机

的特性无关，因此可以简化生成中间代码的编译程序，也便于移植到其他机器和对其进行优化处理。比较常用的中间代码形式有逆波兰表示法、N-元式表示法和图表示法。逆波兰表示法中，每个操作符直接跟在其操作数后面，这样可以省去表达式中的括号，使其变得更加简洁，同时也使运算的处理更加方便；N-元式表示法中，每条指令由 N 个域组成，第一个域通常表示操作符，其余 N-1 个域表示操作数或中间及最后结果，最常用的 N-元式表示法是三元式和四元式；图表示法描述了源程序的自然层次结构，其常见形式是语法树，一个三元式可以对应语法树中一棵二叉子树，而语法树的后序遍历就可以产生逆波兰表达式。

1.3.5 代码优化

代码优化是指在不改变程序运行效果的前提下，对被编译的程序进行变换，是之能生成时间效率和空间效率上更加高效的目标代码。优化的目的就是为了使程序的运行速度提高，存储空间减小，并且在优化的过程中应该遵循等价、有效、合算的原则。

由于代码优化的涉及面很广，因此优化的方法和技术非常多，对优化技术的分类也各不相同。下面介绍一些常用的优化技术分类。

按照优化涉及的源程序的范围，可以把优化分为局部优化、循环优化和全局优化。局部优化是在基本块上实现的优化，基本块是有唯一入口和出口的线性程序序列；循环优化是在程序中循环体范围内的优化；全局优化是在非线性程序块上实现的优化，需要分析该程序块和其他相关程序块，以及整个程序的控制流和数据流。

按照优化相对于编译过程的阶段，可以把优化分为中间代码级上的优化和目标代码级上的优化。中间代码级上的优化与目标机环境无关，包括局部优化、循环优化等；目标代码级上的优化与目标机相关，包括寄存器优化、窥孔优化、并行分支的优化等。

按照优化的具体实现方法，可以把优化分为以下内容：

(1)常量合并与传播。如一段线性程序为“*x=10;y=x+5;*”，则可以优化为“*x=10;y=15;*”。

(2)公共子表达式删除。如一段线性程序为“*c = a + b + 10; d = a + b + 15;*”，则可以优化为“*m = a + b; c = m + 10; d = m + 15;*”。

(3)无用赋值的删除。如一段线性程序为“*a = 7; a = 10;*”，则可以优化为“*a = 10;*”。

(4)死代码删除。如一段程序为

```
char ch;
if(ch > 300) i = 0;
else i = 1;
```

则可以优化为

```
char ch;
i= 1;
```

因为对于字符变量，“ch>300”为永假条件，则 i= 0;永远不会被执行，成为“死代码”。

(5)无用转移语句删除。如一段程序为

```
if(x)
L1: goto L2;
else goto L1;
```

则可以优化为

```
goto L2;
```

(6)循环不变量或不变代码外提。如一段代码为

```
b = a + 10;
for(i = 0; i < N; i++)
   arr[i] = b * 2;
```

则可以优化为

```
b = a + 10;
c = b * 2;
for(i = 0; i < N; i++)
   arr[i] = c;
```

(7)函数内嵌。对于十分简单的函数可以直接嵌入到调用出，以减少函数调用的开销。如一段代码为

```
int foo(int x)
{  return (x == 10); }

void main()
{ if(foo(y)) a = 10; }
```

则可以优化为

```
void main()
{ if(y == 10) a = 10;
```

(8)循环转移。循环优化根据不同的循环结构和定义，会涉及各种优化情况。比较普遍的做法是，将循环内的乘法运算通过某些变换，使之成为循环内的加法运算，降低运算强度，从而达到优化的目的。

(9)其他优化。除了上述一些通用的优化方法外，有时候可以考虑目标机体系结构的特性进行优化。例如对于具有 RISC 处理器的目标机，由于其一般都使用了多阶段流水线的体系结构设计思想，因此可以进行流水线结构的优化，即指令调度。

尽管优化的方法多种多样，但实现优化技术的分析和变换程序相对比较稳定。通常情况下，代码优化阶段可以分为控制流分析、数据流分析和代码变换三部分。其中，控制流分析基于程序流图，通过对程序的扫描和分析识别出程序中的循环部分；数据流分析采集源程序的数据流信息并将其分配给程序流图

的基本块;代码变换则是通过分析中间代码以及控制流分析和数据流分析得到的信息来进行优化。

1.4 本文主要工作

本文主要内容包括绪论、编译器框架设计、编译器框架实现以及编译器框架测试。

本文的主要工是研究一个能够用于教学的可扩展的编译器框架。对于教学用途，是为了让学生们自己写出编译器的各个模块，然后将其模块放入此框架中，从而实现编译器的功能。为了实现这一目的，本文针对编译器的不同过程设计了不同的模块，并开放对学生的模块接口，使其能够通过这些接口，将自己编写的编译器处理模块整合到程序的编译器框架中，从而完成编译的功能。此外,为了让学生对此框架有更好的了解,还在框架中集成了一个内嵌编译器，学生们可以通过内嵌编译器对编译器中文法支持的源程序进行完全的编译运行过程，从而更加清晰地了解每一步编译过程的处理情况，以便能够正确地实现模块接口的功能，编写出自己的模块程序。

1.5 论文组织

本文的第 1 部分是绪论，叙述了课题的研究背景、常见编译器的体系结构、常用的编译器具体实现技术、本文的主要工作和论文组织。

第 2 部分是编译器框架设计说明，包括框架使用方法和运行环境说明、框架的整体架构、内嵌编译器的结构以及编译器的示范和教学用途功能。

第 3 部分是编译器框架实现说明,包括框架的实现方法和内嵌编译器的实现方法。

第 4 部分是编译器框架测试说明,包括内嵌编译器测试和框架试运行情况测试。

第 5 部分是总结，总结了编译器框架的整体情况以及其优势和劣势，并提出了对此编译器框架进行扩展的几点方向。

1.6 小结

本章主要介绍了本文的研究背景、编译器的常用结构和常用的编译器实现技术，同时也阐明了本文所做的主要工作和文章的结构组织。

第 2 章 编译器框架设计

本文所研究的编译器框架针对编译的不同过程设计了不同的模块，具体体现在代码中就是设计了不同的包（Package），每个包中使用一些类来实现功能。最终提交的程序将开放对学生的模块接口，学生根据这些接口标准自己编写编译器处理模块，然后将其放到编译器框架中的对应模块完成编译过程。为了实现对学生的用例示范以及完成程序的默认处理配置，最终提交的程序将包括各个模块的默认处理方式，以 jar 包的形式进行调用。这样做的好处是不仅可以为学生示范标准的接口调用形式、标准的模块文件输入输出形式，还可以将用例程序进行封闭，使其以一个黑盒的方式供学生调用，但学生无法看到程序的源代码，从而可以写出自己风格的代码，达到教学中要求学生自己编程的目的。

2.1 使用方法和运行环境

编译器框架以 jar 包的形式发布，通过命令行调用，调用时需要传入一个命令行参数，代表需要处理的 C 源程序的路径。

框架基于 Java 语言开发，所以框架的运行需要 Java 运行时环境(JRE)。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。

2.2 框架整体架构

2.2.1 框架特点

本文研究的编译器框架是用 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。

2.2.2 框架结构

图 2- 1 显示出本文所研究的编译器框架的基本结构：

此外，对于每个阶段输入输出文件的命名格式，也有明确的定义，具体定义在框架程序的 bit.minisys.minicc 包的 MiniCCCfg.java 文件中。表 2- 1 则是对 MiniCCCfg.java 文件中命名格式定义的表格化表示，并辅以简单的功能描述。

表 2- 1 输入输出文件命名规则及功能描述

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

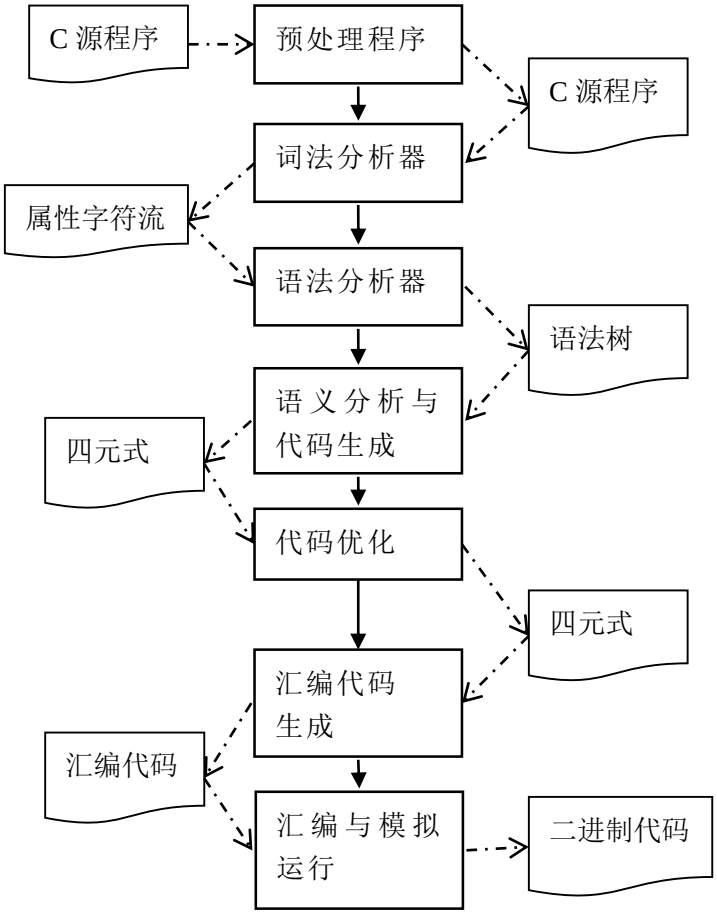

图 2- 1 编译器框架结构

2.3 内嵌编译器结构

内嵌编译器完全按照框架接口而设计，其中包含：

(1)总控程序（bit.minisys.minicc 包）、

(2)预处理子程序（bit.minisys.minicc.pp 包）、

(3)词法分析器（bit.minisys.minicc.scanner 包）、

(4)语法分析器（bit.minisys.minicc.parser 包）、

(5)语义分析子程序（bit.minisys.minicc.semantic 包）、

(6)中间代码生成子程序（bit.minisys.minicc.icgen 包）、

(7)代码优化子程序（bit.minisys.minicc.optimizer 包）、

(8)目标代码生成子程序（bit.minisys.minicc.codegen 包）、

(9)模拟运行子程序（bit.minisys.minicc.simulator 包）

(10)附加工具调用（bit.minisys.minicc.util 包）。

其中，语义分析和代码优化部分在内嵌编译器中并未实现，因为这两部分不影响编译过程的正常实现，故留给学生自行实现。内嵌编译器的结构如图 2-2 所示。

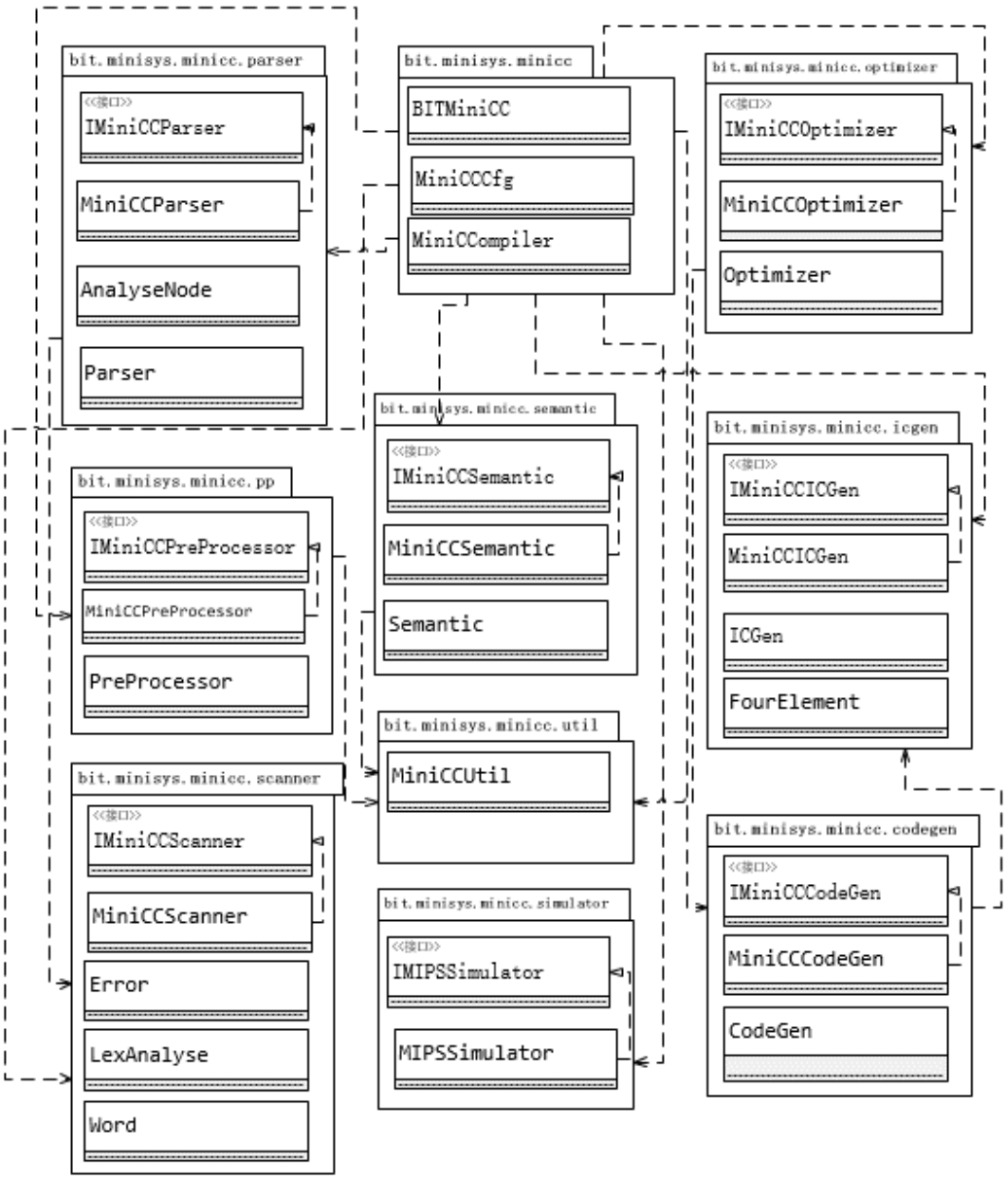

图 2- 2 内嵌编译器类图结构

2.4 示范功能

为了使学生能够更加直观地体验编译过程，并且能够在实现流程中较为靠后的编译模块不需要考虑前面流程的实现，本框架在每个模块内部集成了该模块的功能实现程序，用来进行示范。

在程序代码中，向学生公开的框架代码只是总控程序(bit.minisys.minicc 包)的代码，其余各个包内部的类均不公开，只公开各个包内部需要学生实现的接口，如图 2- 3 所示。

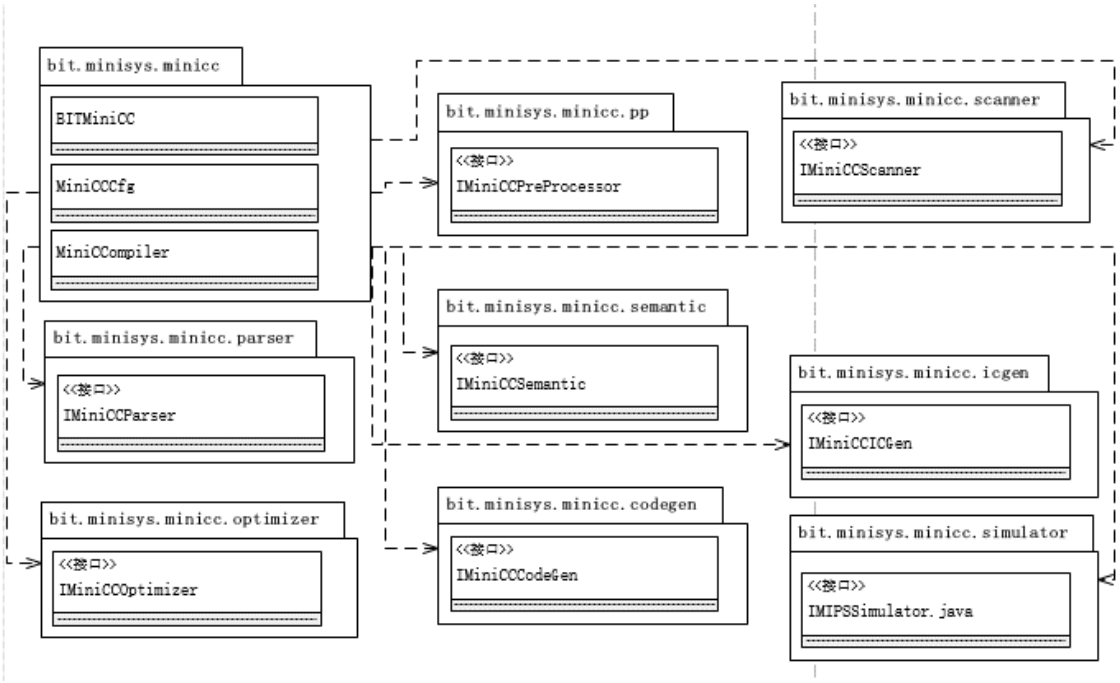

图 2- 3 编译器框架代码组织结构

在总控程序中，BITMiniCC.java 是程序的入口，MiniCCCfg.java 记录的输入输出文件的命名格式，MiniCCompiler.java 负责具体调用每个模块的模块处理子程序，从而实现编译过程。在示范中，进行模块处理的子程序在引用的外部 jar 包中，如图 2- 4 所示。

图 2- 4 编译器框架引用外部 jar 包

在实际操作中，学生可以通过修改程序目录下的 config.xml 文件来控制需要运行那些模块子程序，使用那个子程序来对内嵌编译器模块进行替换等。config.xml 文件中各行标签的含义如表 2- 2 所示：

表 2- 2 config.xml 文件说明

标签名	标签含义及说明
name	表示配置的模块名称
type	表示处理这一模块所用的程序类型，如”java”表示使用 java 源代码或 jar 包，”python”表示使用 python 脚本，”binary”表示使用 C/C++/C# 语言程序（xxx.exe）
path	表示使用 type 所示程序类型时，程序所在的路径。当使用 java 源代码时，本项为空
skip	表示对 name 所示模块，程序运行时是否跳过。如 skip = “true”, name = “parsing”，表示程序运行过程中跳过 parser 阶段

一个 config.xml 文件示例如图 2- 5 所示。

```
<?xml version="1.0" encoding="UTF-8"?>
- <config name="config.xml">
  - <phases>
    - <phase>
        <phase name="pp" path="" type="java" skip="false"/>
        <phase name="scanning" path="" type="java" skip="false"/>
        <phase name="parsing" path="./bin/parse.exe" type="binary" skip="false"/>
        <phase name="semantic" path="" type="java" skip="false"/>
        <phase name="icgen" path="" type="java" skip="false"/>
        <phase name="optimizing" path="./bin/opt.py" type="python" skip="false"/>
        <phase name="codegen" path="" type="java" skip="false"/>
        <phase name="simulating" path="" type="java" skip="true"/>
      </phase>
    </phases>
  </config>
```

图 2- 5 config.xml 文件内容

为了实现模块的功能，可以直接在源代码中添加相应的类和方法，实现每个模块的接口要求即可；或者也可以使用 java 以外的语言，实现模块功能并导出为可执行程序，通过修改配置文件 config.xml 的内容调用可执行程序，从而完成模块功能。可执行程序的调用形式为：xxx iFileName oFileName，即使用 xxx.exe 或 python 脚本作为模块子程序时，需要接受两个命令行参数，第一个是模块子程序要处理的输入文件名，第二个是模块子程序处理后的输出文件名。

2.5 教学用途功能

本框架在教学中的使用方法，是学生按照规定的接口编写模块子程序，所编写的子程序需要满足接口的要求，然后通过总控程序调用相应的内嵌子程序和自己编写的模块子程序，就可完成编译过程。

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。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，并修改 config.xml 文件，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。

2.5 小结

本章主要介绍了编译器框架的使用方法和运行环境、框架的整体架构、内嵌编译器的结构以及框架的两个主要功能：示范功能和教学用途功能。其中，教学用途功能只需将模块中的内嵌子程序隐藏起来（生成 jar 包）并引用生成的 jar 包即可，这样不仅使内嵌编译器源程序对学生透明，同时也有利于学生集中精力解决任意模块的问题。

第 3 章 编译器框架实现

本文研究的编译器框架的可扩展性体现在对框架的模块接口进行扩展，使其不仅能够处理默认形式的 jar 包或 Java 源代码，还可以处理诸如 C 语言程序的 exe 形式，或者 Python 脚本等模块实现方式，从而实现多种语言的支持。这样设计的好处在于，不需要学生掌握某一门独特的语言，只需要能够按照标准的接口和输入输出文件格式，写出满足程序要求的模块，就可以放入程序框架中运行，降低了学生实现编译过程在语言方面的门槛，提高了程序的可扩展性。此外，为了使学生们使用编译器框架时能够直观地看到编译过程的各个部分，同时也能够集中精力处理编译过程的某一个或某几个模块而不受其他模块是否实现的影响，本框架内部集成了一个内嵌编译器。此编译器完全符合框架的结构和接口设计，即由一系列子程序组成，并分别处理对应的不同编译过程的子模块。学生在使用本框架时，可以选取任意的模块进行编程实现，只需用自己所实现的模块子程序将框架内置的模块子程序替换，框架就会在运行时调用相应模块的学生编写的子程序，大大方便了学生对某一个或某几个模块的实现。下面就编译器框架及框架内嵌编译器的具体实现加以说明。

3.1 框架实现方法

框架实现代码为 bit.minisys.minicc 包（即总控程序）内的代码，包内类的组织结构如图 3- 1 所示。

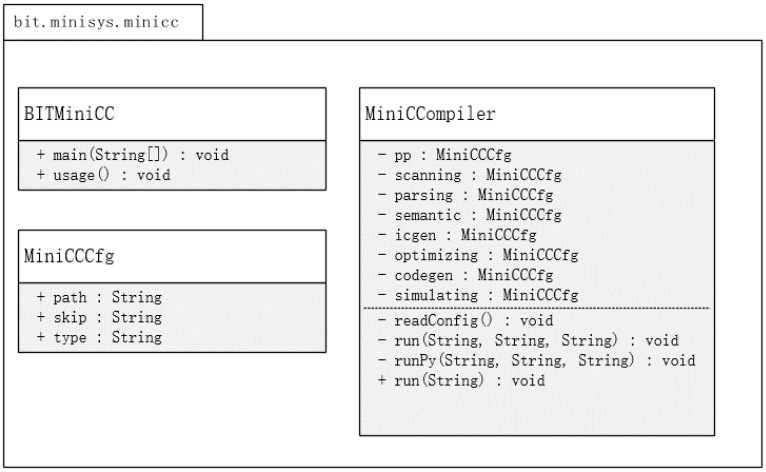

图 3- 1 总控程序代码组织结构

其中，BITMiniCC.java 是程序的入口，MiniCCCfg.java 规定了文件命名规则，MiniCComplier.java 是总控程序实现程序，负责调用各个模块的接口类，实现编译过程。相关类的具体介绍如下：

1) BITMiniCC.java

此类仅处理输入的 C 语言文件，判断输入文件是否符合格式，如果符合则调用 MiniCComplier.java 中的 run()方法进行编译处理。

2) MiniCCCfg.java

此类仅定义文件命名格式，其代码定义如表 3- 1 所示。

表 3- 1 MiniCCCfg.java 定义文件命名格式

变量名	代表后缀	代表含义
MINICC_PP_INPUT_EXT	.c	C 源程序
MINICC_PP_OUTPUT_EXT	.pp.c	删除无用注释和空格，宏替换与文件包含
MINICC_SCANNER_INPUT_EXT	.pp.c	预处理过的 C 程序
MINICC_SCANNER_OUTPUT_EXT	.token.xml	词法分析，生成属性字符流
MINICC_PARSER_INPUT_EXT	.token.xml	词法分析后的属性字符流
MINICC_PARSER_OUTPUT_EXT	.tree.xml	语法分析，生成语法树
MINICC_SEMANTIC_INPUT_EXT	.tree.xml	语法树
MINICC_SEMANTIC_OUTPUT_EXT	.tree2.xml	语义检查
MINICC_ICGEN_INPUT_EXT	.tree2.xml	语法树
MINICC_ICGEN_OUTPUT_EXT	.ic.xml	生成四元式列表
MINICC_OPT_INPUT_EXT	.ic.xml	中间代码
MINICC_OPT_OUTPUT_EXT	.ic2.xml	实施常量合并等代码优化
MINICC_CODEGEN_INPUT_EXT	.ic2.xml	中间代码
MINICC_CODEGEN_OUTPUT_EXT	.code.s	生成 x86 或者 MIPS 汇编代码
MINICC_ASSEMBLER_INPUT_EXT	.code.s	目标代码

3) MiniCComplier.java

此类为总控程序实现程序，通过对 config.xml 的内容进行判断，从而决定如何对每个模块进行处理。

在编译器框架中，总控程序调用的类名是固定的，因此学生自己编写程序实现模块功能时，必须将类名定义为固定的名字。然而，这样做降低了框架的灵活性。为了解决这一问题，框架中采用了 Java 中的“反射”机制，使得学生可以不需要将类名固定为规定的名字就可以使用框架调用自己的模块功能实现程序。

反射机制主要是指程序可以访问、检测和修改它本身状态或行为的一种能力，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。通俗来讲，反射机制的使用使得在程序运行过程中，可以知道任意一个类的所有属性和方法，也可以调用任意一个对象的所有属性和方法。总而言之，使用反射机制就可以在程序运行过程中获取自身的诸多信息。

总控程序具体实现算法如 3 - 1 所示

```
算法 3-1：
read(config.xml)，update(moduleList);       //moduleList 代表所有模块变量
for each module in moduleList：
    if module.skip == false then
        if module.type.equals(“java”) then
            if module.path != “” then                                    //反射机制
                Class<?> c = Class.forName(module.path);
                Method method = c.getMethod();
                method.invoke(c.newInstance(), inFile, outFile);
            else
                runJava();
            end if
        else if modul.type.equals(“python”) then
                runPython();
            end if
        else
            runExe();
        end if
    end if
end for
```

3 - 1 总控程序实现算法

3.2 内嵌编译器实现方法

3.2.1 预处理

内嵌编译器的预处理子程序功能十分简单，只是将源程序中的一些注释删除。预处理子程序的代码组织情况如图 3- 2 所示。

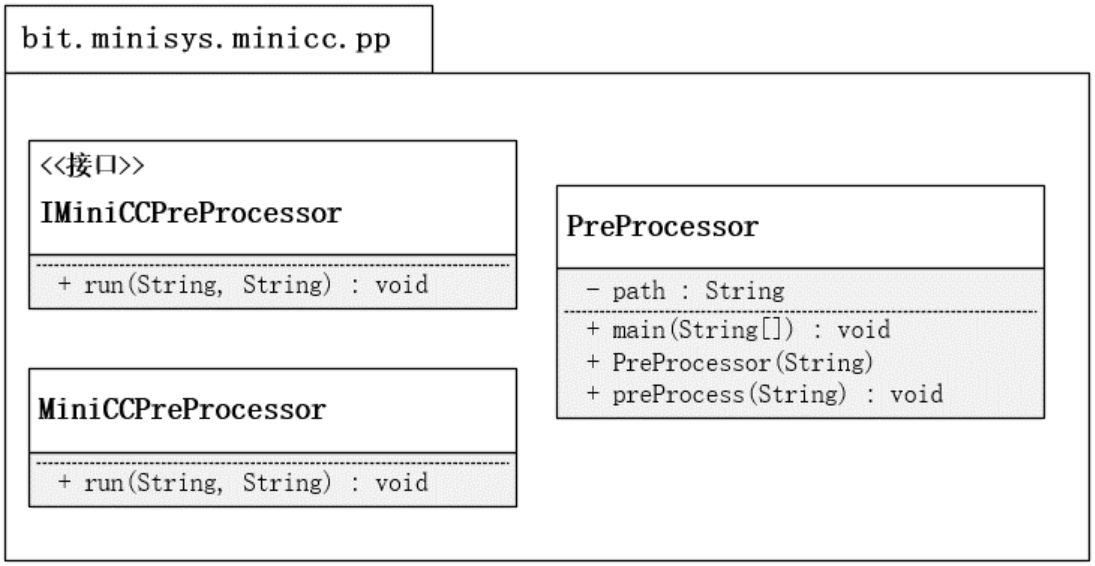

图 3- 2 预处理模块代码结构

其中，IMiniCCPreProcessor.java 是预处理模块接口，规定了预处理模块必须实现的方法；MIniCCPreProcessor.java 是子程序入口，由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用；PreProcessor.java 是预处理模块实现程序，在内嵌编译器中，其主要功能是删去文件中的注释，核心代码如 3 - 2 所示。

```
算法 3-2：
checkFilePath(); readFile();
for each line in file do
    if find(“//” || “/**/”); then delete(line);
    end if
    output(line);
end for
```

3 - 2 预处理模块算法

3.2.2 词法分析

词法分析器的实现以状态转换图为基础，在内嵌编译器的词法分析模块中，以程序中心法对状态转换图进行实现，即将各类状态转换图进行合并，构成一个能识别所有单词的状态转换图之后，使用条件判断语句对输入串进行判断，从而识别出合法的单词。

编译器框架的单词（终结符）编码如表 3- 2 所示。

词法分析器的代码结构如图 3- 3 所示

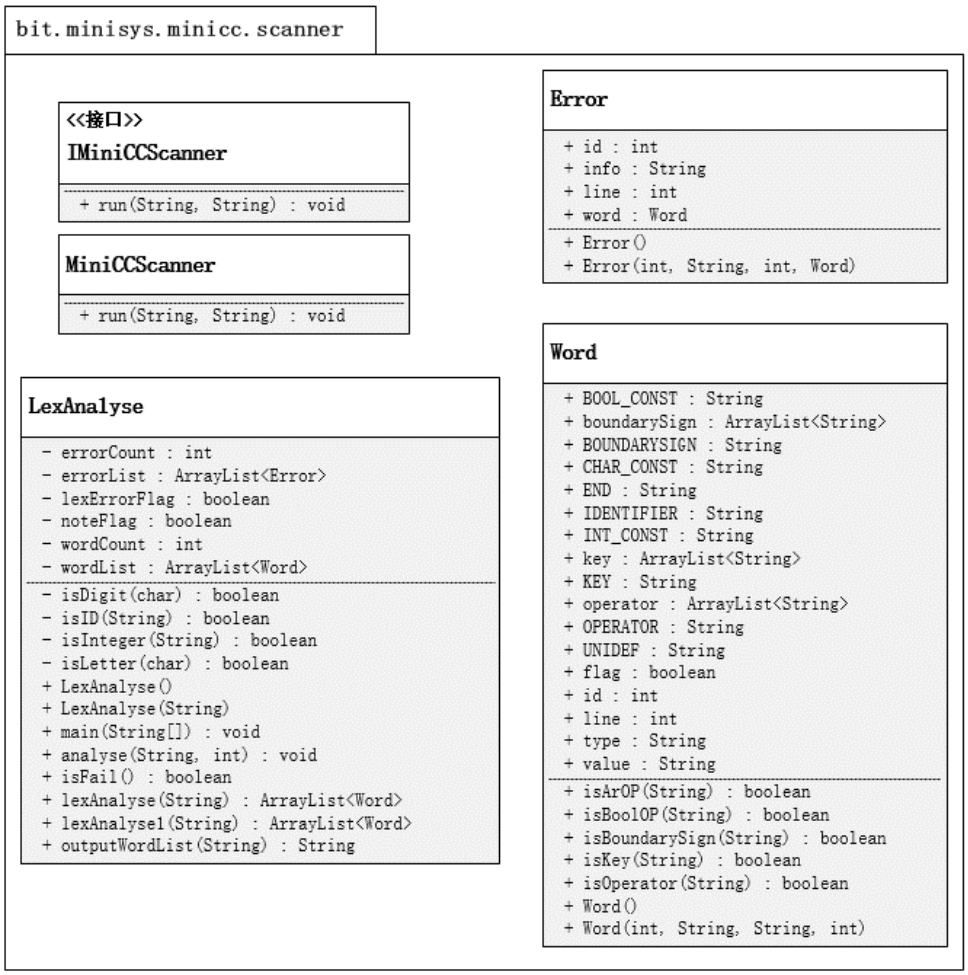

图 3- 3 词法分析器代码结构

表 3- 2 内嵌编译器 BITMiniCC 终结符编码

类别	类型	编码值	内码值（举例）
关键字 keyword	int	0x00010001	
	char	0x00010002	
	float	0x00010003	
	void	0x00010004	
	if	0x00010005	
	else	0x00010006	
	for	0x00010007	
终结符 identifier	id	0x00020001	i
常量 const	整型常量 const_i	0x00030001	25
	字符常量 const_c	0x00030002	a
	浮点型常量 const_f	0x00030003	3.5
界限符 separator	{	0x00040001	{
	}	0x00040002	}
	(	0x00040003	(
	)	0x00040004	)
	;	0x00040005	;
	,	0x00040006	,
运算符 operator	+	0x00050001	+
	-	0x00050002	-
	++	0x00050003	++
	--	0x00050004	--
	*	0x00050005	*
	>	0x00050006	>
	<	0x00050007	<
	>=	0x00050008	>=
	<=	0x00050009	<=
	=	0x00050010	=
	==	0x00050011	==
	!=	0x00050012	!=
	&&	0x00050013	&&
	‖	0x00050014	‖
	!	0x00050015	!
	|	0x00050016	|
	&	0x00050017	&

其中，IMiniCCScanner.java 是词法分析器接口，规定了词法分析器必须实现的方法；MIniCCScanner.java 是子程序入口，由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用；LexAnalyse.java 是词法分析器实现程序，完成了对状态转换图的实现，识别出输入串中的单词；Error.java 是错误信息类，规定了错误信息的具体属性值有哪些；Word.java 是单词类，规定了终结符的类型、种类，包含判断终结符的一些方法。其中，较重要的几个类介绍如下。

1) Error.java

此类记录了错误信息的结构，其含义如表 3- 3 所示。

表 3- 3 错误变量说明

变量名	变量含义
id	错误序号
info	错误信息
line	错误所在行
word	错误的单词

2) Word.java

此类定义了单词的结构，如图 3- 3 所示。此外，也定义了词法分析器支持的终结符、关键词、界限符等信息，其含义如表 3- 4 所示。

表 3- 4 单词类变量说明

变量名	变量含义
id	单词序号
value	单词的值
type	单词类型
line	单词所在行
flag	单词是否合法

3) LexAnalyse.java

此类进行词法分析，即单词的判断。类中定义了 wordList 和 errorList 两个 ArrayList 对象，分别存储识别出的单词和词法分析过程中出现的错误信息。分析过程中，根据单词的状态转换图编写条件判断语句，进行识别。识别完成后，形成属性字符流，以 XML 文件的形式输出。

XML 文件输出需要调用 jdom.jar 外部包进行操纵，需要先声明根节点和一个 Document 对象，将根节点放入 Document 对象中，然后创建节点，为节点赋予属性并将其加入到根节点中，最后输出 Document 对象即可。具体算法如 3 - 3 所示。

```
算法 3-3：
Element root = new Element().setAttribute();          // 创建根节点 并设置它的属性
Document Doc = new Document(root);                    // 将根节点添加到文档中
Element tokens = new Element("tokens");               // 创建 tokens 节点并添加到 root
root.addContent(tokens);
for each word in wordlist do
    Element elements = new Element("token");          // 创建节点 token
    elements.addContent();                            // 给 tokens 节点添加子节点并赋值
    tokens.addContent(elements);                      // 把节点添加到 root 中
end for
Format format = Format.getPrettyFormat();             // 定义输出格式
XMLOutputter XMLOut = new XMLOutputter(format);
XMLOut.output(Doc, new FileOutputStream(output));     // 输出文件
```

3 - 3 XML 文件输出算法

3.2.3 语法分析

语法分析是在词法分析的基础上，读入词法分析输出的属性字符流，按照文法规则进行处理，生成语法树。本框架的内嵌编译器采用 LL(1)分析方法进行语法分析。

LL(1)分析法是由一个总控程序读入输入字符串，通过查 LL(1)分析表，在一个分析栈上运行而完成语法分析的任务的。LL(1)分析法由总控程序、LL(1)分析表和分析栈组成，其逻辑结构如图 3- 4 所示。

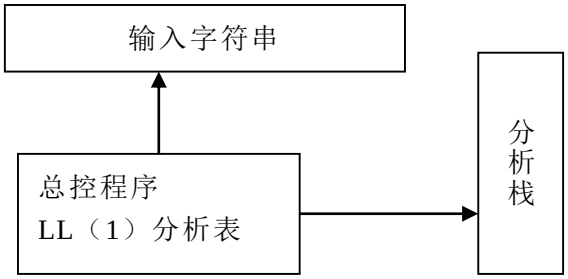

图 3- 4 LL(1)分析器逻辑结构

LL(1)分析表以 M[A,a]形式的矩阵表示，其中 A 表示文法的非终结符，a 表示文法的终结符或'#'（结束符）。每一个矩阵元素存放一个非终结符 A 的候选式或空白。在[A,a]中的矩阵元素为 X，则表示分析栈顶的非终结符为 A、读入的字符为 a 时，应该用 A→X 这个产生式进行推导。

分析栈中存放语法分析过程中的文法符号。初始化时在栈底压入'#'号表示结束符，然后压入文法开始符号。

总控程序通过分析当前栈顶符号和读入的字符来确定下一步的动作，从而进行分析。总控程序的算法如 3 - 4 所示：

到此为止，LL(1)分析法只剩下分析表的构造方法没有解决。在构造 LL(1)分析表时，需要两个概念，即 FIRST 集合和 FOLLOW 集合。

设文法 G 是 2 型文法，则文法 G 中$\forall\beta \in V^*$的终结首符集 FIRST(β)为：

$$\mathrm{FIRST}(\beta) = \{a|\beta \overset{*}{\Rightarrow} a\ldots,\ a \in V_T\} \qquad (3\text{-}1)$$

若$\beta \overset{*}{\Rightarrow} \varepsilon$，则$\varepsilon \epsilon \mathrm{FIRST}(\beta)$。

设上下文无关文法（3 型文法）G，S 是文法的开始符号，对于文法 G 的任何非终结符 A 的 FOLLOW 集 FOLLOW(A)为：

$$FOLLOW(A) = \{a|S \overset{*}{\Rightarrow} \ldots Aa\ldots,\ a \in V_T^*\} \qquad (3\text{-}2)$$

若$S \overset{*}{\Rightarrow} \ldots A$，则$\# \in \mathrm{FOLLOW}(A)$。

有了 FIRST 集合和 FOLLOW 集合的定义，LL(1)分析表就可以构造出来，构造算法如 3 - 5 所示：

```
算法 3-4：
push ‘#’, S                                                    // S 是文法开始符号
read top, firstWord;                  //top 是栈顶符号，firstWord 是读入的字符
if isTerminal(top) and isTerminal(firstWord) then
    if top == ‘#’ and firstWord == ‘#’ then
        succeed;
    else if top == firstWord and top != ‘#’ then
            pop top;
        end if
    else if top != firstWord then
            error;
        end if
    end if
else
    if M[top, firstWord] is null then          // M[top, firstWord]储存产生式
        error;
    else
        pop top;
        reverse_push M[top, firstWord];
    end if
end if
```

3 - 4 LL(1)分析总控程序算法

```
算法 3-5：
for each Ai → r1|r2|…|rm do
    if a∈FIRST(ri), then
        M[Ai,a] = Ai→ri;
    end if
    if ε∈FIRST(ri) then
        for each a∈FOLLOW(Ai) do
            M[Ai,a] = Ai→ri;
        end for
    end if
end for
```

3 - 5 LL(1)分析表构造算法

使用 LL(1)分析法进行语法分析，需要采用 LL(1)文法。本框架的内嵌编译器采用的文法是 C 语言文法的子集，经过简单修改成为 LL(1)文法，其文法规则定义如图 3- 5 所示。

对图 3- 5 中文法符号说明如表 3- 5 语法分析器文法符号说明所示：

表 3- 5 语法分析器文法符号说明

非终结符		终结符	
文法符号	符号含义	文法符号	符号含义
PROGRAM	程序	TKN_ID	标识符
FUNCTIONS	函数集	TKN_CONST_I	整型常量
FUNLIST	函数列表	TKN_LP	左括号”(”
FUNCTION	函数	TKN_RP	右括号”)”
ARGS	函数参数	TKN_COMMA	逗号”,”
ALIST	函数参数列表	TKN_LB	左大括号”{”
FARGS	函数参数实现	TKN_RB	右大括号”}”
FUNC_BODY	函数体	TKN_SEMICOLON	分号”;”
STMTS	语句集	TKN_KW_RET	返回关键字”return”
STMT	语句	TKN_FOR	for 关键字”for”
EXPR_STMT	表达式语句	TKN_IF	if 关键字”if”
EXPR	表达式	TKN_ELSE	else 关键字”else”
FACTOR	表达式实现	TKN_INT	int 关键字”int”
FLIST	表达式列表	TKN_FLOAT	float 关键字”float”
EARGS	变量定义	TKN_ASN	赋值号”=”
EALIST	变量定义列表	TKN_PLUS	加法运算符”+”
RET_STMT	返回语句	TKN_LESS	小于号运算符”<”
FOR_STMT	for 循环语句	ε	空
IF_STMT	if 条件判断语句		
ELSEIF	else if 型判断语句		
ILIST	if 列表		
TYPE	变量类型		

```
PROGRAM → FUNCTIONS
FUNCTIONS → FUNCTION FUNLIST
FUNLIST → FUNCTIONS || ε

FUNCTION → TYPE TKN_ID TKN_LP ARGS TKN_RP FUNC_BODY
ARGS → FARGS ALIST || ε
ALIST → TKN_COMMA FARGS ALIST || ε
FARGS → TYPE TKN_ID

FUNC_BODY → TKN_LB STMTS TKN_RB
STMTS → STMT STMTS || ε
STMT → EXPR_STMT || RET_STMT || FOR_STMT || IF_STMT

EXPR_STMT → EXPR TKN_SEMICOLON
EXPR → FACTOR FLIST || EARGS
FACTOR → TKN_LP EXPR TKN_RP || TKN_ID
FLIST → TKN_ASN FACTOR FLIST || TKN_PLUS FACTOR FLIST ||
TKN_LESS FACTOR FLIST || ε
EARGS → FARGS EALIST || ε
EALIST → TKN_COMMA TKN_ID EALIST || ε

RET_STMT → TKN_KW_RET EXPR_STMT

FOR_STMT →TKN_FOR TKN_LP EXPR_STMT EXPR_STMT EXPR
TKN_RP TKN_LB STMTS TKN_RB

IF_STMT → TKN_IF TKN_LP EXPR TKN_RP TKN_LB STMTS TKN_RB
ELSEIF
ELSEIF → TKN_ELSE ILIST || ε
ILIST →IF_STMT || TKN_LB STMTS TKN_RB

TYPE → TKN_INT || TKN_FLOAT
```

图 3- 5 内嵌编译器文法规则

语法分析器的代码结构如图 3- 6 所示：

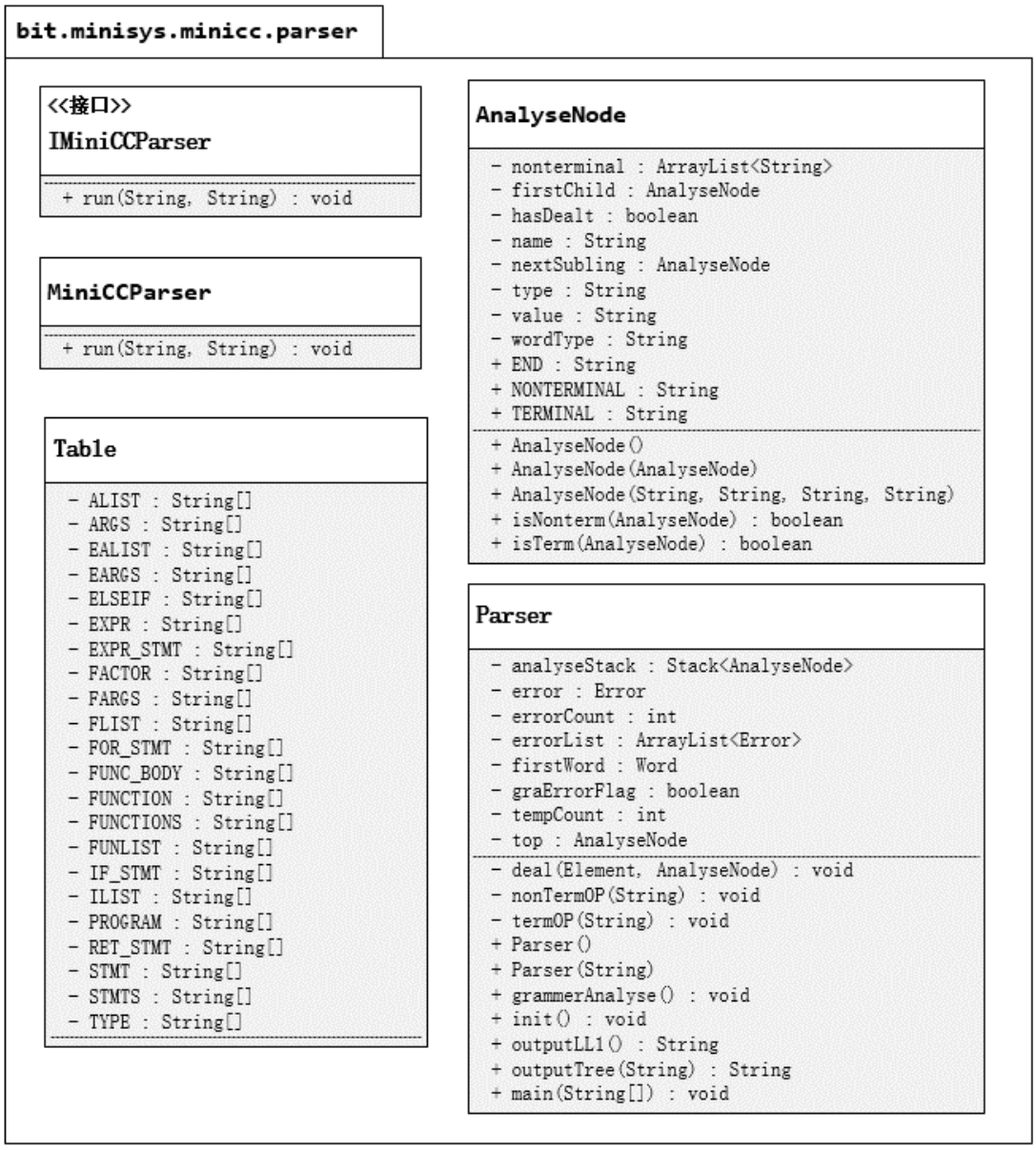

图 3- 6 语法分析器代码结构

其中，IMiniCCParser.java 是语法分析器接口，规定了语法分析器必须实现的方法；MIniCCSParser.java 是子程序入口，由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用；Parser.java 是语法分析器实现程序，对词法分析器输出的属性字符流进行语法处理，生成语法树；Table.java 记录语法分析

器使用的文法；AnalyseNode.java 是节点类，包含语法树节点的属性以及判断文法符号类型的方法。对其中较为核心的类介绍如下。

1) AnalyseNode.java

此类定义了语法树节点的结构，其含义如表 3- 6 节点类变量所示。

表 3- 6 节点类变量说明

变量名	变量含义
type	节点类型
name	节点名
value	节点值
firstChild	语法树子节点
nextSubling	语法树兄弟节点
hasDealt	标志是否在 XML 文件中处理过
wordType	节点单词的类型

2) Parser.java

此类实现了语法分析，采用的方法为 LL(1)分析法。

类中定义了所有的终结符与非终结符节点，其定义如图 3- 7 所示。

```
AnalyseNode PROGRAM, FUNCTIONS, FUNLIST, FUNCTION, ARGS, ALIST, FARGS, FUNC_BODY, STMTS, STMT,
            EXPR_STMT, RET_STMT, FOR_STMT, IF_STMT, EXPR, FACTOR, FLIST, EARGS, EALIST,
            ELSEIF, ILIST, TYPE;//非终结符
AnalyseNode TKN_ID, TKN_CONST_I, TKN_LP, TKN_RP, TKN_COMMA, TKN_LB, TKN_RB, TKN_SEMICOLON, TKN
            TKN_PLUS, TKN_MINUS, TKN_LESS, TKN_DIV, TKN_ASN, TKN_INT, TKN_FLOAT,
            TKN_FOR, TKN_IF, TKN_ELSE;//终结符
```

图 3- 7 非终结符/终结符定义

语法分析时首先需要读取词法分析的属性字符流 XML 文件，利用 org.w3c.dom 读取，其具体算法如 3 - 6 所示。

```
算法 3-6：
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(filePath);
NodeList wordList = doc.getElementsByTagName("name");
for each element in wordlist do
    for each child in element do
        if node.getNodeType() == Node.ELEMENT_NODE then
            child.getNodeValue();
        end if
    end for
end for
```

3 - 6 读取 XML 文件算法

读入 XML 文件属性字符流之后，就可开始语法分析，具体算法如 3 - 7 所示。

在语法分析过程中，每进行一次非终结符节点的处理，就将其产生式节点均置为该节点的子节点。当所有属性字处理完毕时，从文法开始符号出发，即可遍历所有处理过的节点，按照这些节点之间的父子关系，可以构造出一棵语法树。

语法分析完成后，以 XML 形式生成语法树，其算法与 3 - 3 类似。

```
算法 3-7：
push ‘#’, S                                                    // S 是文法开始符号
read top, firstWord;                  //top 是栈顶符号，firstWord 是读入的字符
if top == ‘#’ and firstWord == ‘#’ then
    succeed;
else if top == ‘#’ then
    error;
else if top is terminal then
    if top == firstWord then
        pop top;
    end if
else if top is nonterminal then
        if M[top, firstWord] is null then      // M[top, firstWord]储存产生式
            error;
        else
            pop top;
            reverse_push M[top, firstWord];
        end if
    end if
end if
```

3 - 7 语法分析器实现算法

3.2.4 中间代码生成

中间代码生成需要对程序语句进行翻译，根据语句的类型不同，中间代码生成的算法也不同。在内嵌编译器中，中间代码由四元式表示，以 XML 文件的形式输出。

四元式的形式为：(OP, ARG1， ARG2， RESULT)，其中 OP 表示操作符，如表达式中的+、-等运算符和判断或循环语句中的 J、JT 等跳转指令；ARG1、ARG2 表示第一、第二操作数，在某些四元式中可以省略；RESULT 表示运算结果，用一个变量或临时变量来表示，内嵌编码器中临时变量用 Ti 来表示。

内嵌编译器中的中间代码生成模块代码组织结构如图 3- 8 所示。

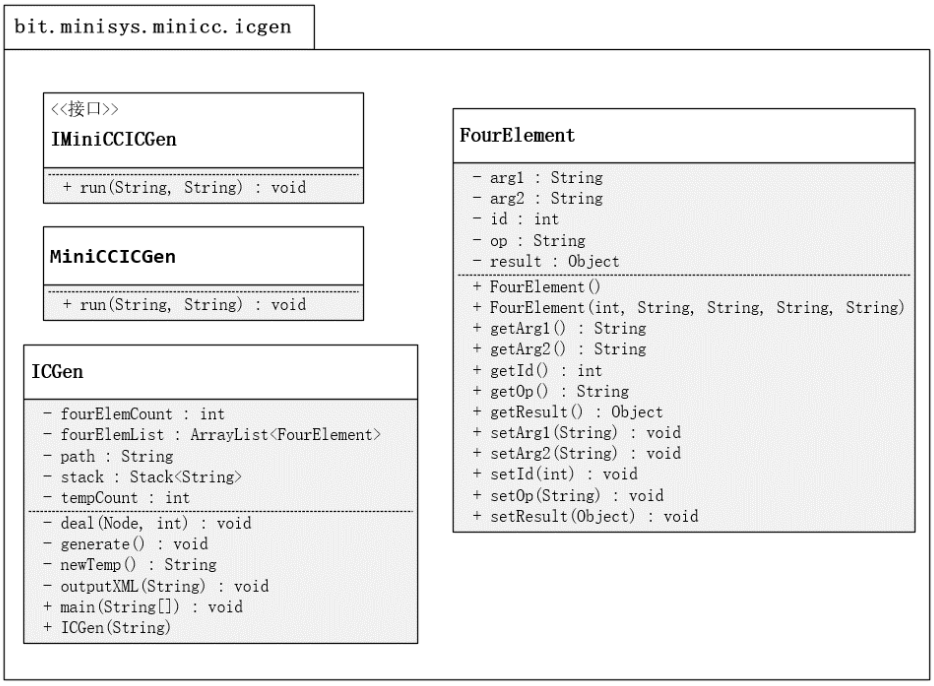

图 3- 8 中间代码生成模块代码结构

其中，IMiniCCICGen.java 是中间代码生成模块的接口，规定中间代码生成模块必须实现的方法；MIniCCICGen.java 是子程序入口，由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用；ICGen.java 是中间代码生成模块实现程序，实现了从语法树到中间代码的转换过程；FourElement.java 是四元式类，定义了四元式的结构。其中较为重要的类介绍如下。

1) FourElement.java

此类定义了四元式的结构，其含义如表 3- 7 所示。

表 3- 7 四元式变量说明

变量名	变量含义
id	四元式序号
op	操作符
arg1	第一操作数
arg2	第二操作数
result	运算结果

2) ICGen.java

此类实现了中间代码生成，其算法如 3 - 8 所示

```
算法 3-8：
read tree; get nodeList;
for each node in nodeList do
    push node and subtree;
    while stack is not empty do
        pop elements;
        judge and generate code
    end while
end for
```

3 - 8 中间代码生成算法

3.2.5 目标代码生成

目标代码生成需要读入中间代码，即四元式，然后对四元式向目标代码进行翻译。内嵌编译器最终翻译的目标代码是 MIPS 汇编代码，其代码结构如图 3- 9 所示。

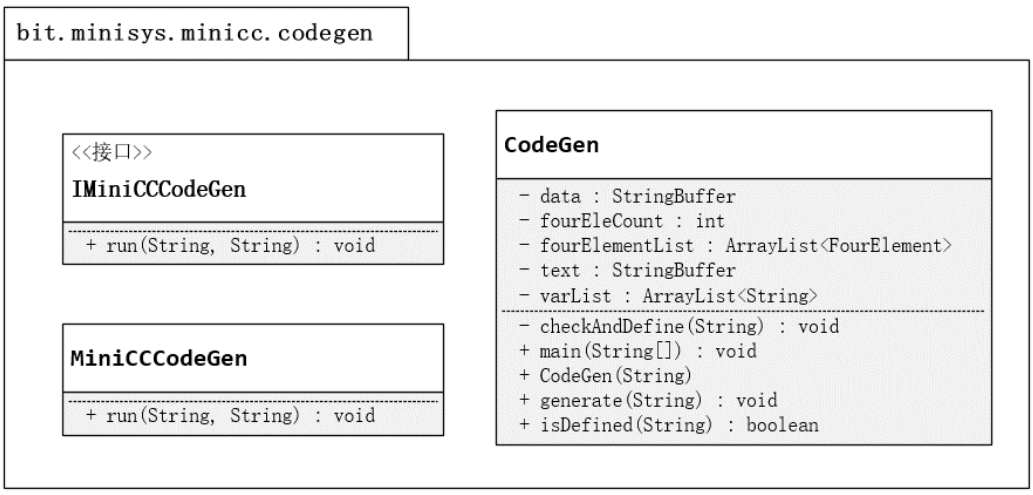

图 3- 9 目标代码生成模块代码结构

其中，IMiniCCCodeGen.java 是目标代码生成模块的接口，规定目标代码生成模块必须实现的方法；MiniCCCodeGen.java 是子程序入口，由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用；CodeGen.java 是目标码生成模块实现程序，读入中间代码的四元式形式，并将其翻译为 MIPS 汇编代码。

CodeGen.java 生成目标代码的算法如 3 - 9 所示。

```
算法 3-9：
read fourElementFile, get fourElementList;
for each element in fourElementList
    chechAndDefine(variety);
    allocate register;
    generate code;
end for
```

3 - 9 目标代码生成算法

3.2.6 模拟运行

模拟运行模块集成了 MARS4.5 进行汇编和模拟运行，使用 MAR4.5 需要在 http://courses.missouristate.edu/KenVollmar/MARS/下载 MARS4.5jar 包，然后在项目中导入外部 jar 包即可。

模拟运行模块的代码组织结构如图 3- 10 所示。

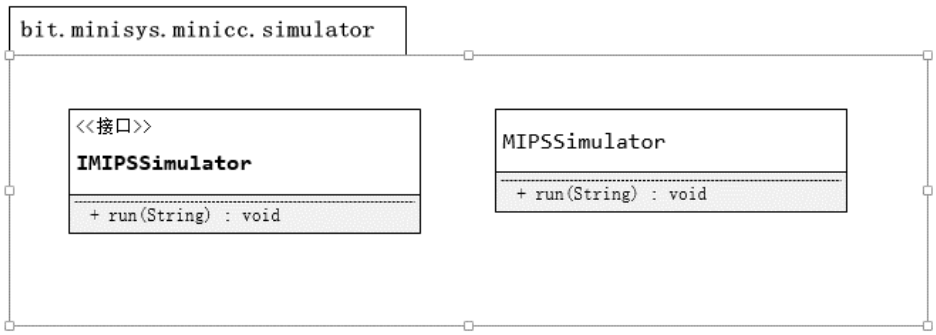

图 3- 10 模拟运行模块代码结构

其中，IMIPSSimulator.java 是模拟运行模块的接口，规定模拟运行模块模块必须实现的方法；MIPSSimulator.java 是模拟运行实现程序，通过调用外部 jar 包 MARS4.5，打开 MARS4.5 的界面，然后进行汇编和模拟运行，这部分由总控程序 bit.minisys.minicc 中的 MiniCCompiler.java 调用。

3.3 小结

本章主要介绍了内嵌编译器的相关情况，包括内嵌编译器的组成结构，以及组成编译器的各个模块的代码组织结构、输入输出文件格式说明等，包括预处理、词法分析、语法分析、中间代码生成、目标代码生成、模拟运行几个模块。

第 4 章 编译器框架测试

编译器框架完成后，对其进行了一系列测试，主要分为两部分，即对框架内嵌编译器的测试以及对框架试运行情况的测试。

4.1 内嵌编译器测试

内嵌编译器完成后，对内嵌编译器进行功能测试，由于编译器实现的模块化，只需完整处理一次 C 语言源程序就可以获得各个阶段的结果，因此，采用不同组织结构的 C 语言源程序进行以下测试分析。

4.1.1 仅表达式的程序

首先测试文法支持的最简单程序，即仅包含表达式的程序，源程序即各模块测试结果如图 4- 1 到图 4- 7 所示。

```
int main(int a, int b){
//a simple example
    j = a + b;
    return a + b;
}
```

图 4- 1 测试源程序 1

```
int main(int a, int b){
    j = a + b;
    return a + b;
}
```

图 4- 2 预处理结果 1

```
<?xml version="1.0" encoding="UTF-8"?>
- <project name="test.l">
  - <tokens>
    - <token>
        <number>1</number>
        <value>int</value>
        <type>keyword</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>2</number>
        <value>main</value>
        <type>identifier</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>3</number>
        <value>(</value>
        <type>separator</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>4</number>
        <value>int</value>
        <type>keyword</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>5</number>
        <value>a</value>
        <type>identifier</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>6</number>
        <value>,</value>
        <type>separator</type>
        <line>1</line>
        <valid>true</valid>
      </token>
```

图 4- 3 词法分析结果 1

```
<?xml version="1.0" encoding="UTF-8"?>
- <ParserTree name="test.tree.xml">
  - <PROGRAM>
    - <FUNCTIONS>
      - <FUNCTION>
        - <TYPE>
            <keyword>int</keyword>
          </TYPE>
          <identifier>main</identifier>
          <separator>(</separator>
        - <ARGS>
          - <FARGS>
            - <TYPE>
                <keyword>int</keyword>
              </TYPE>
              <identifier>a</identifier>
            </FARGS>
          - <ALIST>
              <separator>,</separator>
            - <FARGS>
              - <TYPE>
                  <keyword>int</keyword>
                </TYPE>
                <identifier>b</identifier>
              </FARGS>
              <ALIST/>
            </ALIST>
          </ARGS>
          <separator>)</separator>
        - <FUNC_BODY>
            <separator>{</separator>
          - <STMTS>
            - <STMT>
              - <EXPR_STMT>
                - <EXPR>
                  - <FACTOR>
                      <identifier>j</identifier>
                    </FACTOR>
                  - <FLIST>
                      <operator>=</operator>
                    - <FACTOR>
                        <identifier>a</identifier>
                      </FACTOR>
                    - <FLIST>
                        <operator>+</operator>
                      - <FACTOR>
                          <identifier>b</identifier>
                        </FACTOR>
                        <FLIST/>
                      </FLIST>
                    </FLIST>
                  </EXPR>
```

图 4- 4 语法分析结果 1

```
<?xml version="1.0" encoding="UTF-8"?>
- <IC name="test.ic.xml">
  - <functions>
    - <function>
        <quaternion addr="1" op="+" arg1="a" arg2="b" result="T1"/>
        <quaternion addr="2" op="=" arg1="" arg2="T1" result="j"/>
        <quaternion addr="3" op="+" arg1="a" arg2="b" result="T2"/>
        <quaternion addr="4" op="return" arg1="" arg2="T2" result=""/>
      </function>
    </functions>
</IC>
```

图 4- 5 中间代码生成结果 1

```
.data
T1: .word
a: .word
b: .word
j: .word
T2: .word
.text
.globl main
main:
    la $a0, T1
    la $t1, a
    la $t2, b
    add $t3, $t1, $t2
    sw $t3, 0($a0)
    la $a0, j
    la $v1, T1
    sw $v1, 0($v0)
    sw $v0, 0($a0)
    la $a0, T2
    la $t1, a
    la $t2, b
    add $t3, $t1, $t2
    sw $t3, 0($a0)
```

图 4- 6 目标代码生成结果 1

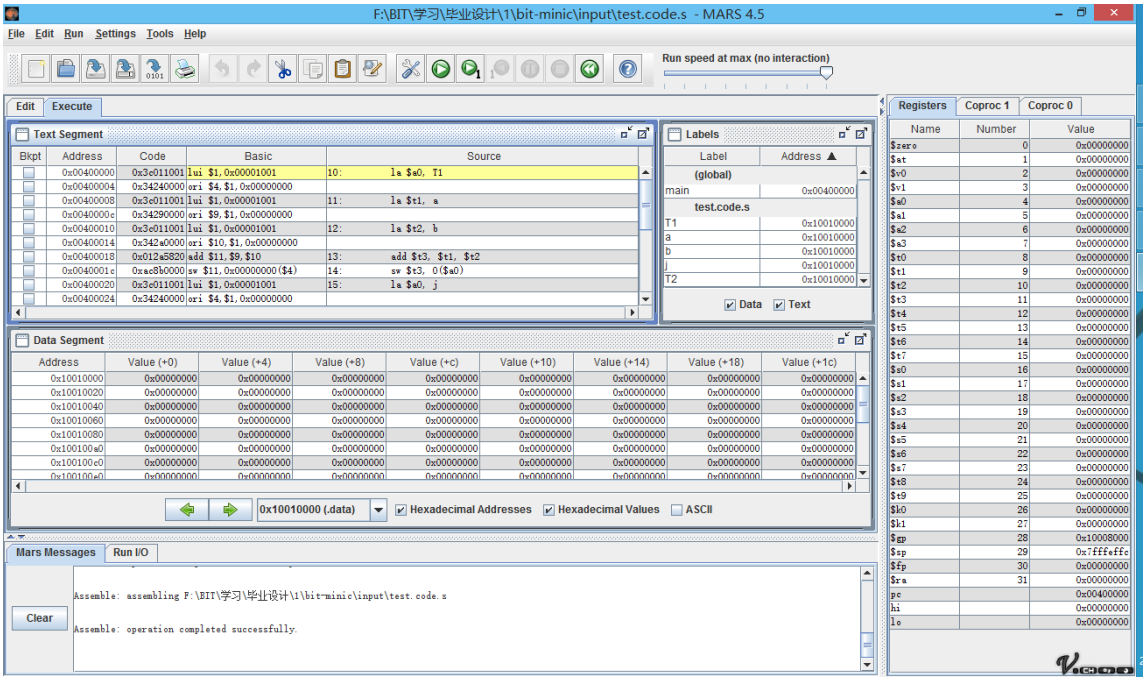

图 4- 7 模拟运行结果 1

由上图可以看出，对于只有表达式的简单 C 语言源程序，所有模块均可正常运行，即可完成编译器的全部功能。

4.1.2 包含所有文法支持语句的多函数程序

第二项测试为包含内嵌编译器文法支持的所有语句的 C 语言源程序，其中包含表达式、返回语句、if/else 循环、for 循环、多函数。测试结果如图 4- 8 到图 4- 12 所示。

```
int main(int a, int b){
//a simple example
        int i, j;
        j = a + b;
        for(i = 0; i < 10; i = i + 1){
                if(a < b){
            a = b;
       }else{
            b = a;
       }
        }
        return a + b;
}
int foo(int c, int d){
//a simple example
        int i, m;
        for(i = 0; i < 10; i = i + 1){
                if(c < d){
            c = d;
       }else{
            d = c;
       }
        }
        m = c + d;
        return m;
}
```

图 4- 8 测试源程序 2

```
int main(int a, int b){
        int i, j;
        j = a + b;
        for(i = 0; i < 10; i = i + 1){
                if(a < b){
            a = b;
       }else{
            b = a;
       }
        }
        return a + b;
}
int foo(int c, int d){
        int i, m;
        for(i = 0; i < 10; i = i + 1){
                if(c < d){
            c = d;
       }else{
            d = c;
       }
        }
        m = c + d;
        return m;
}
```

图 4- 9 预处理结果 2

```
<?xml version="1.0" encoding="UTF-8"?>
- <project name="test.l">
  - <tokens>
    - <token>
        <number>1</number>
        <value>int</value>
        <type>keyword</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>2</number>
        <value>main</value>
        <type>identifier</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>3</number>
        <value>(</value>
        <type>separator</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>4</number>
        <value>int</value>
        <type>keyword</type>
        <line>1</line>
        <valid>true</valid>
      </token>
    - <token>
        <number>5</number>
        <value>a</value>
        <type>identifier</type>
        <line>1</line>
        <valid>true</valid>
      </token>
```

图 4- 10 词法分析结果 2

```
<?xml version="1.0" encoding="UTF-8"?>
- <ParserTree name="test.tree.xml">
  - <PROGRAM>
    - <FUNCTIONS>
      - <FUNCTION>
        - <TYPE>
            <keyword>int</keyword>
          </TYPE>
          <identifier>main</identifier>
          <separator>(</separator>
        - <ARGS>
          - <FARGS>
            - <TYPE>
                <keyword>int</keyword>
              </TYPE>
              <identifier>a</identifier>
            </FARGS>
          - <ALIST>
              <separator>,</separator>
            - <FARGS>
              - <TYPE>
                  <keyword>int</keyword>
                </TYPE>
                <identifier>b</identifier>
              </FARGS>
              <ALIST/>
            </ALIST>
          </ARGS>
          <separator>)</separator>
        - <FUNC_BODY>
            <separator>{</separator>
          - <STMTS>
            - <STMT>
              - <EXPR_STMT>
                - <EXPR>
                  - <EARGS>
                    - <FARGS>
                      - <TYPE>
                          <keyword>int</keyword>
                        </TYPE>
                        <identifier>i</identifier>
                      </FARGS>
                    - <EALIST>
                        <separator>,</separator>
                        <identifier>j</identifier>
                        <EALIST/>
                      </EALIST>
                    </EARGS>
                  </EXPR>
```

图 4- 11 语法分析结果 2

```
<?xml version="1.0" encoding="UTF-8"?>
- <IC name="test.ic.xml">
  - <functions>
    - <function>
        <quaternion addr="1" op="," arg1="i" arg2="j" result="T1"/>
        <quaternion addr="2" op="int" arg1="" arg2="T1" result=""/>
        <quaternion addr="3" op="+" arg1="a" arg2="b" result="T2"/>
        <quaternion addr="4" op="=" arg1="" arg2="T2" result="j"/>
        <quaternion addr="5" op=")" arg1="+" arg2="{" result="T3"/>
        <quaternion addr="6" op="i" arg1="=" arg2="T3" result="T4"/>
        <quaternion addr="7" op="i" arg1="(" arg2="=" result="T7"/>
        <quaternion addr="8" op="for" arg1="" arg2="T7" result=""/>
        <quaternion addr="9" op="{" arg1=")" arg2="}" result="T8"/>
        <quaternion addr="10" op="b" arg1="<" arg2="T8" result="T9"/>
        <quaternion addr="11" op="a" arg1="(" arg2="T9" result="T10"/>
```

图 4- 12 中间代码生成结果 2

由上图可知，预处理、词法分析、语法分析部分处理结果均正确，中间代码生成部分出错。出错的原因是目前还没有支持 if/else、for 循环等跳转语句的中间代码生成部分，需要在以后加以改进。

4.1.3 单词错误的程序

本次测试编译器的词法分析器出错处理程序，将一个变量名改为非法，进行测试，测试结果如图 4- 13 到图 4- 15 所示。

```
int main(int a, int b){
//a simple example
        int 3h;
}
```

图 4- 13 测试源程序 3

```
int main(int a, int b){
        int 3h;
}
```

图 4- 14 预处理结果 3

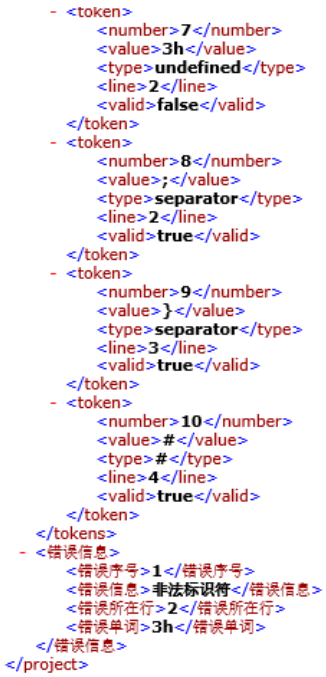

图 4- 15 词法分析结果 3

由上图可知，词法分析器检查出了单词的非法，即“3h”这个词的类型 type 为“undefiend”。同时在 XML 文件的末尾，指出了错误的单词，及其错误序号、错误信息和错误所在行。

4.1.4 语法错误的程序

本次测试编译器的语法分析器出错处理程序，错误信息将在 LL1.log 文件中显示。测试代码将一条语句改为文法并不支持的函数调用，进行测试，如图 4- 16 到图 4- 20 所示。

```
int main(int a, int b){
//a simple example
        j = foo();
}
```

图 4- 16 测试源程序 4

```
int main(int a, int b){
        j = foo();
}
```

图 4- 17 预处理结果 4

```
            <valid>true</valid>
        </token>
      - <token>
            <number>9</number>
            <value>(</value>
            <type>separator</type>
            <line>2</line>
            <valid>true</valid>
        </token>
      - <token>
            <number>10</number>
            <value>)</value>
            <type>separator</type>
            <line>2</line>
            <valid>true</valid>
        </token>
      - <token>
            <number>11</number>
            <value>;</value>
            <type>separator</type>
            <line>2</line>
            <valid>true</valid>
        </token>
      - <token>
            <number>12</number>
            <value>}</value>
            <type>separator</type>
            <line>3</line>
            <valid>true</valid>
        </token>
      - <token>
            <number>13</number>
            <value>#</value>
            <type>#</type>
            <line>4</line>
            <valid>true</valid>
        </token>
    </tokens>
</project>
```

图 4- 18 词法分析结果 4

```
<?xml version="1.0" encoding="UTF-8"?>
- <ParserTree name="test.tree.xml">
  - <PROGRAM>
    - <FUNCTIONS>
      - <FUNCTION>
        - <TYPE>
            <keyword>int</keyword>
          </TYPE>
          <identifier>main</identifier>
          <separator>(</separator>
          <ARGS/>
          <separator>)</separator>
        - <FUNC_BODY>
            <separator>{</separator>
          - <STMTS>
            - <STMT>
              - <EXPR_STMT>
                - <EXPR>
                  - <FACTOR>
                      <identifier>j</identifier>
                    </FACTOR>
                  - <FLIST>
                      <operator>=</operator>
                    - <FACTOR>
                        <identifier>foo</identifier>
                      </FACTOR>
                      <FLIST/>
                    </FLIST>
                  </EXPR>
                  <separator/>
                </EXPR_STMT>
              </STMT>
              <STMTS/>
            </STMTS>
            <separator>}</separator>
          </FUNC_BODY>
        </FUNCTION>
        <FUNLIST/>
      </FUNCTIONS>
    </PROGRAM>
  </ParserTree>
```

图 4- 19 语法分析结果 4

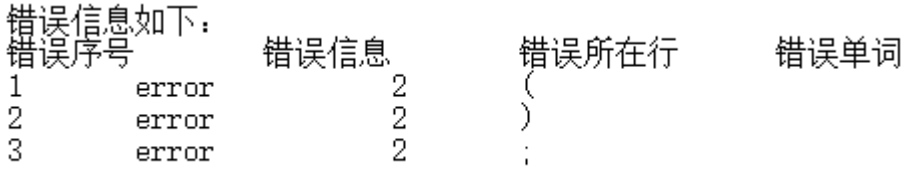

图 4- 20 语法分析错误信息

由上图可知，语法分析错误处理程序检查出语法错误，即在程序第二行单词为’(‘, ‘)’, ‘;’处出错。

4.2 编译器框架试运行情况测试

本文所研究的编译器框架供北京理工大学计算机学院学生《编译原理课程》使用，当框架初步完成时，就上线供学生们下载用试用，至今已获得一批学生的使用情况反馈。在试用过程中，框架整体运行较为稳定，但仍出现了一些问题，将在下文进行一一介绍。

4.2.1 使用方式

学生可以在 Github 上下载编译器框架，下载地址为 https://github.com/jiweixing/bit-minic-compiler。框架以 jar 包的形式发布，同时学生也可以直接下载教学用框架源码，即只有框架源码以及各模块接口，各模块具体实现以 jar 包形式调用。运行框架是，需要通过通过命令行调用，调用时需要传入一条参数，代表需要处理的 C 源程序的路径。程序会将 C 源程序进行处理，处理的步骤有：预处理、词法分析、语法分析、语义检查、中间代码生成、代码优化、目标代码生成、模拟运行。每个阶段处理完成后，都会生成相应的文件，文件存储目录与 C 源程序所在目录相同。

4.2.2 使用中发现的问题

在框架试运行的过程中，学生在试用了框架之后，发现了一系列问题，先列举如下：

1) 从 Github 上下载框架之后，在本机上运行出错，无法正常打开文件，如图 4- 21 所示。

```
<已终止> BITMiniCC [Java 应用程序] C:\Users\a0312\Desktop\jre1.8.0_45\bin\javaw.exe ( 2016-4-27 下午9:56:53 )
USAGE: BITMiniCC FILE_NAME.c
请在控制台中输入源文件
D:\工程项目\java\BIT-MiniCC-Clean\src\bit\minisys\minicc\test\test.c
Start to compile ...
1. PreProcess finished!
2. LexAnalyse finished!
Exception in thread "main" java.net.MalformedURLException: unknown protocol: d
        at java.net.URL.<init>(Unknown Source)
        at java.net.URL.<init>(Unknown Source)
        at java.net.URL.<init>(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLEntityManager.setupCurrentEntity(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLVersionDetector.determineDocVersion(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.DOMParser.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl.parse(Unknown Source)
        at javax.xml.parsers.DocumentBuilder.parse(Unknown Source)
        at bit.minisys.minicc.parser.Parser.<init>(Parser.java:68)
        at bit.minisys.minicc.parser.MiniCCParser.run(MiniCCParser.java:11)
        at bit.minisys.minicc.MiniCCompiler.run(MiniCCompiler.java:119)
        at bit.minisys.minicc.BITMiniCC.main(BITMiniCC.java:43)
```

图 4- 21 本机无法运行框架

2) 在使用框架的过程中，内嵌编译器的词法分析结果会使属性字的“行号”属性均为“1”，无法找到错误发生的具体位置。
3) 使用内嵌编译器时，一些简单的语句如赋值语句无法识别，语法分析及其后续步骤会出错。
4) 学生在使用框架时，发现实现了各模块的接口以后，如果类的命名与总控程序中调用的类名不同，则无法正常调用学生自己的模块程序，十分不方便。

4.2.3 问题的解决方案

针对 4.2 中学生反馈的问题，对编译器框架进行改进，将问题逐一解决，具体解决方案如下：

1) 针对本机无法运行的问题，经检查后发现读取 XML 文件的路径（URL）问题，XML 文件的 URL 不可包含英文，故将文件的路径改为纯英文即可正常运行框架。
2) 针对“行号”属性均为“1”的问题，经检查发现是预处理时将源程序中的回车换行符全部删除，整个源代码被处理为一个长字符串，即所有代码均在一行，因此会出现行号均为“1”的问题。经过改进，预处

理程序将以 XML 文件形式输出，以便方便保留源程序中每行代码的行数。

3) 针对简单语句无法识别的问题，这是由于最初试运行时内嵌编译器使用的文法过于简单，不支持赋值语句等简单语句。经改进后，内嵌编译器的文法得到了扩充，支持赋值语句、for 循环语句、if/else 条件判断语句、返回语句等，对简单的 C 语言程序可以进行处理。

4) 针对无法调用学生自己编写的模块程序的问题，这是由于总控程序中调用的类名是确定的，如果学生没有按照规定的类命名的话，就会调用外部 jar 包中的内嵌编译器模块子程序。为解决这一方法，在总控程序中引入了“反射”机制，使学生无需完全按照规定命名类也可以运行自己编写的模块程序，大大提高了框架的灵活性。

4.3 小结

本章主要对框架内嵌编译器进行了测试，采用不同结构的 C 语言源代码，测试目前内嵌编译器的功能情况。此外，介绍了编译器框架的使用方法，以及框架在试用过程中发现的问题和对应的解决方案。本文研究的编译器框架最终目的是用于教学中，因此当基本内容完成时，在学生中进行试用是很有必要的，也只有经过实践检验，才能发现更多的问题和不足，以便对框架进行改进，使之更加完善。

第 5 章 总结

《编译原理》课程的学习可以让学生们更加深入地了解计算机程序较为低层的实现机制，与平时使用的高级语言联系起来，能够更加准确、全面地把握计算机的整体运作。为了让学生们能够更加快速、有效的学习并应用编译原理的基本方法，本文研究了一个编译器的框架，此框架可以让学生能够亲自观察编译过程每个模块的具体处理内容，同时也帮助学生在集中精力处理某个模块的编译过程时，能够轻松地看到该模块的处理情况。

本框架分为 8 个模块，包括预处理、词法分析、语法分析、语义分析、中间代码生成、代码优化、目标代码生成、模拟运行。每个模块都可以独立处理一段编译过程，所有模块综合运行就可以完成编译器的整个编译过程。这样设计的好处在于灵活方便，学生可以轻松地处理任意模块而不用担心与其相关的其他模块是否会有问题。

本框架主要有两项功能。一项功能是用作示范，即内部集成了一个简单的编译器，学生可以用来运行、查看每个模块的编译过程，这有利于学生理解、接触编译器的构成和实现方式，让学生更加容易入门。另一项功能是用作教学，即学生需要自己编程实现模块的功能，其他功能模块的内置子程序对学生透明，这样可以激发学生自己动手处理问题的能力，让学生更好地掌握编译原理的基本方法。

此外，编译器框架进行了一段时间的试运行，期间发现了一些问题，也解决了一些问题。总体来讲，经过试运行后，编译器框架得到了改进，与原先相比性能、处理能力和兼容性都有所提高。

目前编译器框架只是初步版本，功能并不强大，而框架的兼容性很高，今后可以从以下几个方面对内嵌编译器进行改进：

1) 增强预处理功能，模块具备处理宏替换、文件包含和条件编译等功能。
2) 改进词法分析器，扩充单词库。
3) 改进语法分析器，目前内嵌编译器使用的语法分析方法为 LL(1)分析法，之后可改为 LR 分析法等能够识别更多语言的分析方法。相应的文法也可进行修改，如加入结构、函数调用等。
4) 增加代码优化模块，对中间代码进行优化，改进编译性能。

总之，尽管编译器框架功能还有很大提升空间，但目前的版本已经可以供学生进行使用，而且框架的功能非常使用，其内置的编译器也能够满足学生使用的一般要求，能够让学生体验到编译过程的各个模块情况。因此，总体来说，本文研究的编译器框架能够满足预期的要求，是一次成功的研究。

致　谢

春华秋实，时光飞逝，不经意间已经度过了大学的四年本科时光。回首以往，刚刚入学的情景还历历在目，细细想来，却已经历了数不尽的风霜雨雪。四年以来，求学路上有过欢笑，有过泪水，有过感慨，有过寂寥，但最终，却不得不迎来一次分别。在此，谨向那些对我有过关心、有过帮助的良师益友致以最真挚的敬意和最深切的感谢。

首先，我要感谢我的毕业设计导师计卫星老师，他严谨的治学态度和积极的工作热情都深深感染了我。一年以来，从毕设课题的选择、研究到最终论文的撰写，计老师都严格要求并给予我深刻的鼓励。在我拿到课题茫然不知如何入手之时，计老师简单几句点拨便让我觉得柳暗花明；在我完成一个阶段的工作之时，计老师总能及时的查收，并提出自己的意见以及和我讨论下一个阶段的工作；在我有时懈怠不专心于课题工作时，计老师也能及时鞭策我，让我能够在预计的时间内完成工作。在这论文撰写结束的时刻，计老师一年以来的关怀一幕幕浮现在我眼前，师恩浩荡，铭记于心，争取成绩，以报师恩。

然后，感谢入学以来一直对我有所帮助的班主任吴心筱老师、辅导员张杨老师、杜东阳老师。是你们从我入学之时就一路引导我，让我熟悉大学的生活，在学校中感受到了家的温暖。

此外，还要感谢和我共同经历四年大学生活的同学们，正是因为有你们，我的大学生活才不会孤独乏味。是你们与我同甘苦、共欢笑，一同陪我度过珍贵的大学时光。毕业在即，一起经历过的日子彷如昨天，共同的欢笑与扶持，共同的拼搏与努力，都是我们之间最宝贵的财富。谢谢你们陪我度过大学时光，让我在最美好的年纪留下最深刻的回忆。

最后，我要感谢我的父母，感谢你们一直以来对我生活的关心，对我学习的支持。相信我，今后一定更加努力，奋勇前行，取得更加优秀的成绩。

参考文献

[1] 陈英，王贵珍，李侃，计卫星，陈朔鹰．编译原理［M］. 北京：清华大学出版社，2009.

[2] Andrew W. Appel, Jens Palsberg. Modern Compiler Implementation In Java, Second Edition［M］. Cambridge：Cambridge University Press，2002.

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers principles, techniques, and tools［M］. New York：Pearson Education, Inc，2007.

[4] 李侃，陈英，计卫星，王贵珍．“编译原理与设计”课程三维一体化教学改革［J］. 计算机教育，2011(14)：38-48.

[5] 陈英，计卫星，王贵珍，李侃．编译原理-课程教学指导思想的思忖［J］. 计算机教育，2009(21).

[6] Joseph Yiu. The Definitive Guide to the ARM Cortex-M0［M］. London：Elsevier Inc，2011. 385-403.

[7] 计卫星，陈英，王贵珍，李侃．编译原理研究型课程实验设计与实践［J］. 计算机教育，2014(18)：99-102.

[8] 陈英，计卫星，王贵珍，李侃．编译原理与设计课程研究型教学模式探讨［J］. 计算机教育，2013(17)：24-26.

[9] 计卫星，陈英，王贵珍，李侃．基于插件的编译原理课程实验设计［J］. 计算机教育，2011(11)：20-27.

[10] Dipnarayan Guha, Thambipillai Srikanthan. Compiler Back End Design for Translating Multi-radio Descriptions to Operating System-less Asynchronous Processor Datapaths［J］. Journal of Computers，2008，3(1)：7-14.

[11] Doug Baldwin. A compiler for teaching about compilers［J］. ACM SIGCSE Bulletin，2003：220-223.

[12] Tyson R. Henry. Teaching compiler construction using a domain specific language［J］. ACM SIGCSE Bulletin，2005(3)：7-11.

[13] 计卫星，陈英，李侃，王贵珍．简析计算机专业知识在编译课程教学中的渗透与融合［J］．计算机教育，2010(3)：23-25.

[14] 李侃，陈英，计卫星，王贵珍．面向编译原理与设计课程的学生创新性思维阶梯式培养模式［J］．计算机教育，2013(17)：11-14.

[15] 王涛，卢军，张凯兵．一种基于图形可视化的编译原理计算机辅助教学系统［J］．湖北工程学院学报，2015(3)：85-88.

[16] 蔡杰．GCC 编译系统结构分析与后端移植实践［D］．浙江：浙江大学，2004.

[17] 史英超. PIM-C 语言设计与实现[D]. 西安：西安理工大学，2011.

[18] 陈慈勇．基于 GCC 的交叉编译器移植开发及其测试方案的设计研究［D］．厦门：厦门大学，2011.

[19] 朱少波．基于 GCC 开发 C 编译器的研究与实践［D］．浙江：浙江大学，2003.

[20] 李海丰．嵌入式 C 编译器优化技术的研究与实现［D］．天津：南开大学，2008.

[21] 王国云．嵌入式软件的编译器优化分析及安全性验证技术研究［D］．浙江：浙江大学，2008.