**目 录**

**第1章 绪论**

1.1 研究背景和意义

1.2 编译器技术的研究现状

1.3 主要研究内容

1.3.1 虚拟机

1.3.2 编译系统

**第2章 虚拟机设计与实现**

高级语言与目标机

2.1 基于寄存器的虚拟机

2.2 基于栈的虚拟机

2.3 指令集设计

2.4 数据的存储

2.5 指令的存储

2.6 数据对齐

2.7 虚拟机实现

2.8 虚拟机测试

**第3章 编译器设计**

3.1 符号表设计

3.1.1 符号的表示

3.1.2 作用域的标识

3.1.3 查找和建立标识符

3.1.4 内建符号的处理

3.2 词法分析

3.2.1词法分析算法的设计

3.2.2 词法单元的识别

3.3 语法分析

3.3.1 语言文法的定义

3.3.2 语法分析算法的设计

3.4 语义分析与代码生成

3.4.1 语法制导的翻译技术

3.4.2 代码生成

**第4章 编译器实现与测试**

4.1 词法分析器的实现

4.1.1 标识符的识别

4.1.2 数字的识别

4.1.3 注释的识别

4.1.4 字符与字符串的识别

4.1.5 其他符号的识别

4.2 语法制导的解析程序实现

4.2.1 变量的解析

4.2.2 enum的解析

4.2.3 函数的解析

4.2.4 语句的解析

4.2.5 编译错误

4.2.6 编译警告

4.3编译器测试

4.3.1 变量测试

4.3.2 enum测试

4.3.3 函数测试

4.3.4 语句的测试

4.3.5 整体测试

**第5章 总结与展望**

5.1 本文总结

5.2 未来展望

**致谢**

**参考文献**

摘要

编译器能将以某种语言写的程序转换为等价的目标语言写成的程序。编译器是计算机科学中理论与实践相结合的最好典范，通过编写编译器，不仅能更好地掌握编程语言本身，还能锻炼组织大量代码的能力、以及系统设计的能力。

编译器通常由一系列阶段组成：词法分析、语法分析、语义分析、代码生成。通过对编译器进行阶段划分，可以让整体结构更加清晰，易于编写和维护。编译器有多趟式编译器和单趟式编译器之分，多趟式编译器通常实现为多个相对独立的阶段，每个阶段都将对源代码进行扫描，生成的目标代码更加高效，但是实现过程更加复杂。单趟式编译器则将编译的四个阶段组合成一遍，只对源代码扫描一次，就完成了编译过程，实现过程更加简单，但是生成的代码没有多趟式编译器生成的代码高效。本文的目的是实现一个能够正确将源代码翻译成目标代码的编译器，因此主要内容集中在编译器前端部分，不生成中间代码，也不进行代码优化，直接生成目标虚拟机指令，采用单趟式实现。

主要的内容总结如下：

（1）定义C语言子集的文法，定义一个实现变量定义、函数定义、各种语句支持的C语言子集。并设计一个虚拟机，其指令作为目标代码，编译器将源代码翻译成虚拟机指令，并启动虚拟机实现目标代码的运行。

（2）采用硬编码方式实现词法分析程序，根据每个单词可能的符号组成，在词法分析程序中直接判断符号，完成词法单元的截取，为后续阶段提供词法单元。

（3）将语法分析、语义分析及代码生成融合在一起进行的程序，语法分析采用自顶向下的LL(1)方法进行语句的解析，语义分析则是简单的检查类型匹配，代码生成部分，则参考《编译原理》中各种语句的代码生成方法直接生成虚拟机指令。

关键字：编译器，虚拟机，单趟式

第一章 绪论

1.1 研究背景和意义

编译器是程序员使用的关键工具，必须生成高效的目标代码，但更重要的是，编译器必须生成正确的目标代码，只有可靠的编译器才能生成可靠的应用程序。编译器本身是一个大而复杂的应用程序。编译器构造原理和技术可以说是计算机科学理论与实践相结合的最好典范。

实现一个编译器，涉及各种编译知识的学习，如词法分析、语法分析、代码生成等，还能更加清晰地了解程序的运行逻辑，例如变量定义时如何存储变量、函数调用时参数传递规则、各种语句的运行方式等。因为编译器是一个比较复杂的程序，在实现编译器的过程中，需要对程序进行模块划分从而让程序具有可扩展性，需要仔细地设计符号表以便能够为编译过程提供完整的符号信息，需要设计栈帧结构以便函数能够准确地进行调用，需要设计优良的目标代码让程序能够更加高效地运行。这非常考验系统设计能力的，也能有效地提升编程能力。

1.2 编译器技术的研究现状

上世纪50年代，IBM的John Backus带领了一个研究小组对FORTRAN语言及其编译器进行开发。由于当时人们对编译理论的了解不多，开发编译器既复杂又艰苦。与此同时Noam Chomsky也开始他对自然语言结构的研究。Chomsky研究出根据语言文法的难易程度以及识别它们所需的算法来对语言分类，即Chomsky架构。它包括了文法的四个层次：0型文法、1型文法、2型文法和3型文法。其中2型文法（上下文无关文法）被证明是程序设计语言中最有用的。解析问题（用于上下文无关文法识别的有效算法）的研究是在60年代和70年代，它相当完善地解决了这个问题，现在它已是编译原理中的一个标准部分。人们接着又深入研究有效生成目标代码的方法，人们通常称为优化技术，并被一直使用至今。

因为编译器作为基础软件，历史十分悠久，很多人都在研究，所以编译器的很多技术已经相当成熟。例如主流的三大C/C++编译器：GCC、Clang、cl，词法分析器生成程序Lex及更加先进的Flex，语法分析器生成程序Yacc以及更加先进的Bison，编译器基础设施框架LLVM等。

GCC（GUN Compiler Collection）是一套由GNU开发的编程语言编译器，是以GPL及LGPL许可证发布的自由软件，现已被大多数操作系统采纳为标准的编译器。GCC 6.0提供了很多新的特性，如OpenMP 4.5、段寄存器支持、目标克隆、扩展存数规则等，完全支持C++14，并支持C++17的实验功能。

LLVM这个名字源于Lower Level Virtual Machine，但这个项目并不局限于创建一个虚拟机，它已经发展成为当今炙手可热的编译器基础框架[4]。LLVM最初以C/C++为编译目标，近年来经过众多机构和开源社区的努力，LLVM已经能够为ActionScript、D、Fortran、Haskell、Java、Objective-C、Swift、Python、Ruby、Rust、Scala等众多语言提供编译支持，而一些新兴语言则直接采用了LLVM作为后端。可以说，LLVM对编译器领域的发展起到了举足轻重的作用。

Lex是一个词法分析器生成器，yacc是一个语法分析器生成器，它们结合使用可以完成编译器前端的大部分工作，生成的LLVM IR由LLVM进行代码优化、机器码生成等，为实现编程语言提供了方便，避免了很多重复的工作。

1.3 主要研究内容

编译器是一个语言翻译程序，能够把源语言书写的程序翻译成等价的目标语言书写的程序。整个翻译过程通常被划分为多个阶段，每个阶段都将源程序的一种表示形式转换成另一种表示形式，并提供给下一个阶段使用。将编译过程划分为多个阶段可以降低实现复杂度，而且每个阶段都可以采用各自适当的方式进行实现而不影响其他阶段，降低各个阶段间的耦合。本编译器的目的是将源代码正确地翻译成目标代码，不考虑代码优化，因此不涉及中间代码生成及代码优化，主要实现了词法分析、语法分析、语义分析及目标代码生成四个阶段。

词法分析是编译过程的第一个阶段，任务是对源程序字符流进行扫描和分解，从而识别出一个个的单词。词法分析方法通常包括基于有限自动机的方式，或硬编码方式。基于有限自动机的方式能够实现词法分析程序的自动构造，可以根据正则表达式动态地更改词法单元描述。硬编码的方式则根据给定词法单元描述直接在程序中硬编码识别，较基于有限自动机的方式更加简单，但是不可动态配置词法单元。

语法分析的作用是识别由词法分析器获取的单词符号序列是否是给定文法的正确句子。目前常用的语法分析方法有自顶向下语法分析方法和自底向上语法分析方法。

自顶向下分析包括确定分析和不确定分析，不确定分析方法是带有回溯的分析方法，实际上是一种穷举的试探方法，因此效率低而代价高。确定分析方法中递归下降方法是常用的语法分析方法，其为每个非终结符编写一个递归过程，每个过程的功能是识别由该非终结符推出的串，其递归调用多且运行速度慢，但是实现较为简单。自底向上分析又包括算符优先分析和LR分析。它们各有优缺点，分别适用于不同的场景。

//TODO: 语义分析、目标代码生成

此外，一个完整的编译器还应该包括符号表管理和错误处理。这两个内容贯穿整个编译流程，编译各个阶段都涉及到构造、查找和更新各种符号数据，同时编译过程中可能发现源程序的各种错误，

需要提供错误的性质以及错误发生位置，以便用户能够快速且方便地定位错误并解决错误。