

<<编译原理>>

图书基本信息

书名：<<编译原理>>

13位ISBN编号：9787040133677

10位ISBN编号：7040133679

出版时间：2003-1

出版时间：高等教育出版社

作者：陈意云

页数：381

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<编译原理>>

前言

本书介绍编译器构造的一般原理、基本设计方法和主要实现技术，可作为高等学校计算机科学及相关专业的教材。

虽然只有少数人从事构造或维护程序设计语言编译器的工作，但是编译原理和技术对高校学生和计算机软件工程技术人员来说仍是重要的基础知识之一。

本书能使读者对程序设计语言的设计和实现有深刻的理解，对和程序设计语言有关的理论有所了解，对宏观上把握程序设计语言来说，能起一个奠基的作用。

本书的学习还有助于读者快速理解、定位和解决在程序调试与运行中出现的问题。

对软件工程来说，编译器是一个很好的实例（基本设计、模块划分、基于事件驱动的编程等），本书所介绍的概念和技术能应用到一般的软件设计之中。

大多数程序员同时也是语言的设计者，虽然只是一些简单语言（如输入输出、脚本语言）的设计者，但学习本书仍有助于提高他们设计这些语言的水平。

编译技术在软件安全、程序理解和软件逆向工程等方面有着广泛的应用。

作为教材，本书有如下一些特点：（1）在介绍语言实现技术的同时，强调一些相关的理论知识，如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。

它们是计算机专业理论知识的一个重要部分，在本书中结合应用来介绍这些知识，有助于学生较快领会和掌握。

（2）在介绍编译器各逻辑阶段的实现时，强调形式化描述技术，并以语法制导定义作为翻译的主要描述工具。

<<编译原理>>

内容概要

本书介绍编译器构造的一般原理和基本实现方法，主要内容包括词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成等。

除了介绍命令式编程语言的编译技术外，本书还介绍面向对象语言和函数式编程语言的实现技术。本书还强调一些相关的理论知识，如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。

本书取材广泛新颖、图文并茂，注意理论联系实际。

本书可作为高等学校计算机科学及相关专业的教材，也可供计算机软件工程技术人员参考使用。

<<编译原理>>

书籍目录

第1章 编译器概述1.1 词法分析1.2 语法分析1.3 语义分析1.4 中间代码生成1.5 代码优化1.6 代码生成1.7 符号表管理1.8 错误诊断和报告1.9 阶段的分组习题1第2章 词法分析2.1 词法记号及属性2.1.1 词法记号、模式、词法单元2.1.2 词法记号的属性2.1.3 词法错误2.2 词法记号的描述与识别2.2.1 串和语言2.2.2 正规式2.2.3 正规定义2.2.4 状态转换图2.3 有限自动机2.3.1 不确定的有限自动机2.3.2 确定的有限自动机2.3.3 NFA到DFA的变换2.3.4 DFA的化简2.4 从正规式到有限自动机2.5 词法分析器的生成器习题2第3章 语法分析3.1 上下文无关文法3.1.1 上下文无关文法的定义3.1.2 推导3.1.3 分析树3.1.4 二义性3.2 语言和文法3.2.1 正规式和上下文无关文法的比较3.2.2 分离词法分析器的理由3.2.3 验证文法产生的语言3.2.4 适当的表达式文法3.2.5 消除二义性3.2.6 消除左递归3.2.7 提左因子3.2.8 非上下文无关的语言结构3.2.9 形式语言鸟瞰3.3 自上而下分析3.3.1 自上而下分析的一般方法3.3.2 LL(1)文法3.3.3 递归下降的预测分析3.3.4 非递归的预测分析3.3.5 构造预测分析表3.3.6 预测分析的错误恢复3.4 自下而上分析3.4.1 归约3.4.2 句柄3.4.3 用栈实现移进一归约分析3.4.4 移进一归约分析的冲突3.5 LR分析器3.5.1 LR分析算法3.5.2 LR文法和LR分析方法的特点3.5.3 构造sLR分析表3.5.4 构造规范的LR分析表3.5.5 构造LALR分析表3.5.6 非LR的上下文无关结构3.6 二义文法的应用3.6.1 使用文法以外的信息来解决分析动作的冲突3.6.2 特殊情况产生式引起的二义性3.6.3 LR分析的错误恢复3.7 分析器的生成器3.7.1 分析器的生成器Yacc3.7.2 用Yacc处理二义文法3.7.3 Yacc的错误恢复习题3第4章 语法制导的翻译4.1 语法制导的定义4.1.1 语法制导定义的形式4.1.2 综合属性4.1.3 继承属性4.1.4 属性依赖图4.1.5 属性计算次序4.2 S属性定义的自下而上计算4.2.1 语法树4.2.2 构造语法树的语法制导定义4.2.3 S属性的自下而上计算4.3 L属性定义的自上而下计算4.3.1 L属性定义4.3.2 翻译方案4.3.3 预测翻译器的设计4.3.4 用综合属性代替继承属性4.4 L属性的自下而上计算4.4.1 删除翻译方案中嵌入的动作4.4.2 分析栈上的继承属性4.4.3 模拟继承属性的计算4.5 递归计算4.5.1 自左向右遍历4.5.2 其他遍历方法4.5.3 多次遍历习题4第5章 类型检查5.1 类型在程序设计语言中的作用5.1.1 引言5.1.2 执行错误和安全语言5.1.3 类型化语言的优点5.2 描述类型系统的语言5.2.1 定型断言5.2.2 定型规则5.2.3 类型检查和类型推断5.3 简单类型检查器的说明5.3.1 一个简单的语言5.3.2 类型系统5.3.3 类型检查5.3.4 类型转换5.4 多态函数5.4.1 为什么要使用多态函数5.4.2 类型变量5.4.3 一个含多态函数的语言5.4.4 代换、实例和合5.4.5 多态函数的类型检查5.5 类型表达式的等价5.5.1 类型表达式的结构等价5.5.2 类型表达式的名字等价5.5.3 记录类型5.5.4 类型表示中的环5.6 函数和算符的重载5.6.1 子表达式的可能类型集合5.6.2 缩小可能类型的集合习题5第6章 运行时存储空间的组织和管理6.1 局部存储分配策略6.1.1 过程6.1.2 名字的作用域和绑定6.1.3 活动记录6.1.4 局部数据的安排6.1.5 程序块6.2 全局存储分配策略6.2.1 运行时内存的划分6.2.2 静态分配6.2.3 栈式分配6.2.4 堆式分配6.3 非局部名字的访问6.3.1 无过程嵌套的静态作用域6.3.2 有过程嵌套的静态作用域6.3.3 动态作用域6.4 参数传递6.4.1 值调用6.4.2 引用调用6.4.3 复写一恢复调用6.4.4 换名调用1习题6第7章 中间代码生成7.1 中间语言7.1.1 后缀表示7.1.2 图形表示7.1.3 三地址代码7.2 声明语句7.2.1 过程中的声明7.2.2 作用域信息的保存7.2.3 记录的域名7.3 赋值语句7.3.1 符号表中的名字7.3.2 临时名字的重新使用7.3.3 数组元素的地址计算7.3.4 数组元素地址计算的翻译方案7.3.5 类型转换7.4 布尔表达式和控制流语句7.4.1 布尔表达式的翻译7.4.2 控制流语句的翻译7.4.3 布尔表达式的控制流翻译7.4.4 开关语句的翻译7.4.5 过程调用的翻译习题7第8章 代码生成8.1 代码生成器设计中的问题8.1.1 目标程序8.1.2 指令选择8.1.3 寄存器分配8.1.4 计算次序选择8.2 目标机器8.2.1 目标机器的指令系统8.2.2 指令的代价8.3 基本块和流图8.3.1 基本块8.3.2 基本块的变换8.3.3 流图8.3.4 下次引用信息8.4 一个简单的代码生成器8.4.1 寄存器描述和地址描述8.4.2 代码生成算法8.4.3 寄存器选择函数8.4.4 为变址和指针语句产生代码8.4.5 条件语句习题8第9章 代码优化9.1 优化的主要种类9.1.1 代码改进变换的标准9.1.2 公共子表达式删除9.1.3 复写传播9.1.4 死代码删除9.1.5 代码外提9.1.6 强度削弱和归纳变量删除9.1.7 优化编译器的组织9.2 流图中

<<编译原理>>

的循环9.2.1 必经结点9.2.2 自然循环9.2.3 前置结点9.2.4 可归约流图9.3 全局数据流分析介绍9.3.1 点和路径9.3.2 到达一定值9.3.3 可用表达式9.3.4 活跃变量分析9.4 代码改进变换9.4.1 公共子表达式删除9.4.2 复写传播9.4.3 寻找循环不变计算9.4.4 代码外提9.4.5 归纳变量删除习题9第10章 编译系统和运行系统10.1 c语言的编译系统10.1.1 预处理器10.1.2 汇编器10.1.3 连接器10.1.4 目标文件的格式10.1.5 符号解析10.1.6 静态库10.1.7 可执行目标文件及装入10.1.8 动态连接10.1.9 处理目标文件的一些工具10.2 Java语言的运行系统10.2.1 Java虚拟机语言简介10.2.2 Java虚拟机10.2.3 即时编译器10.3 无用单元收集10.3.1 标记和清扫10.3.2 引用计数10.3.3 拷贝收集10.3.4 分代收集10.3.5 渐增式收集10.3.6 编译器与收集器之间的相互影响习题10第11章 面向对象语言的编译11.1 面向对象语言的概念11.1.1 对象和对象类11.1.2 继承11.1.3 信息封装11.2 方法的编译11.3 继承的编译方案11.3.1 单一继承的编译方案11.3.2 重复继承的编译方案习题11第12章 函数式语言的编译12.1 函数式程序设计语言简介12.1.1 语言构造12.1.2 参数传递机制12.1.3 变量的自由出现和约束出现12.2 函数式语言的编译简介12.2.1 几个受启发的例子12.2.2 编译函数12.2.3 环境与约束12.3 抽象机的系统结构12.3.1 抽象机的栈12.3.2 抽象机的堆12.3.3 名字的寻址12.3.4 约束的建立12.4 指令集和编译12.4.1 表达式12.4.2 变量的引用性出现12.4.3 函数定义12.4.4 函数应用12.4.5 构造和计算闭包12.4.6 letres表达式和局部变量习题12参考文献

<<编译原理>>

章节摘录

在语言设计的历史上，安全性考虑不足是出于效率上的原因，最典型的是C语言。C以处理低级构造的灵活性著称，它的设计为了保证编程的灵活性而牺牲了安全性，它鼓励在安全的边缘进行编程。

这使得C的程序效率很高，但是也使得它们缺乏安全性。

为了排除程序中的安全漏洞，需要大量的调式工作。

对于安全性不足的语言，可以通过运行时的检查来提高安全性，但是运行检查的代价有时是非常昂贵的。

另外，即使是做了细致静态分析的语言，获得彻底的安全性也需要运行时的代价。

例如，一般来说，数组界检查是不可能完全在编译时删除的。

从另一个角度看，为获得安全性所花的代价是值得的。

安全性使得程序出现执行错误就会停止执行，这可以减少调试的时间。

而且安全性保证运行时结构的完整性，因而使得无用存储单元收集（garbagecollection，俗称垃圾收集，见第10章）可以完成。

而无用存储单元收集大大降低代码开发的时间和代码的规模，虽然它需要一点运行时的代价。

于是，从历史上看，安全和不安全语言之间的选择可能最终与开发时间和执行时间之间的权衡相关。

但是，随着经济和社会越来越依赖于信息技术，关键的信息基础构造——运输、通信、金融市场、能源分配和卫生保健等，都将非常危险地依赖于不是一个管理机构范围内的计算基础和资源。

在我们越来越依赖于这些信息基础构造的同时，提高系统对恶意攻击和有漏洞软件破坏的抵抗能力显得越来越重要。

例如，对于用C编写的系统来说，用缓冲区溢出来对它进行的攻击，已占目前各种攻击的50%，另外还有基于C程序格式串的安全漏洞而进行的攻击等等。

因此，当前在安全和不安全语言之间的选择还与对系统安全性的要求相关。

<<编译原理>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>