编译第一次作业

22373407 王飞阳

1、

- **源程序**：汇编语言或者高级语言的代码。
- **目标程序**：指源程序被编译程序翻译后产生的汇编代码或机械代码。
- **翻译程序**：将汇编语言和高级语言编写的程序翻译成等价的机器语言,实现此功能的程序称为翻译程序。
- **汇编程序**：把汇编语言写的源程序翻译成机器语言的目标程序称为汇编程序。
- **编译程序**：编译程序是将高级语言写的源程序翻译成目标语言的程序。
- **解释程序**：解释程序不是直接将高级语言的源程序翻译成目标程序后再执行,而是一个个语句读入源程序，即边解释边执行。
- **遍**：遍历源程序或中间代码，以产生新的中间代码或目标程序的一次过程。

2、

典型的编译程序可划分为以下几个主要的逻辑部分：

- **词法分析程序**：负责识别程序中的单词，包括保留字、标识符、直接数与运算符等。
- **语法分析程序**：根据语法规则，分析各单词充当的语法成分（如表达式、函数、语句），同时进行正确性检查。
- **语义分析程序**：对各语法成分进行语义分析，生成中间代码（如四元式、三元式、逆波兰式等）。
- **中间代码生成程序**：生成中间代码，这是编译过程中的一个重要步骤，用于将源代码转换为更接近于机器语言的中间形式，以便进行优化。
- **中间代码优化程序**：调整优化中间代码，在等价转换的原则下优化效率，生成更好的目标代码。
- **目标代码生成程序**：由中间代码生成目标程序，此过程中可进行一定优化，如充分利用累加器。
- **表格管理程序**：管理源程序中的各种标识符等。
- **错误处理程序**：识别并定位源程序中的错误。

3、

编译的前端和后端是指编译器中与源语言和目标机相关的不同部分。

- **前端**主要涉及与源语言相关的部分，包括词法分析、语法分析、语义分析与中间代码的产生。这些部分通常与目标机无关，主要处理源代码的解析和转换，生成中间代码或抽象语法树（AST）。前端还包括一些代码优化工作，旨在提高编译效率和生成更优化的中间代码。
- **后端**则主要涉及与目标机相关的部分，包括与目标机有关的代码优化和目标代码生成等。这些部分直接与特定的硬件平台相关，负责将中间代码转换成特定机器可以执行的机器代码。后端的工作还包括进行与目标机相关的优化，以确保生成的机器代码具有最佳的性能和效率。

将编译程序划分为前端和后端的主要目的是为了提高编译系统的开发效率和灵活性。通过将编译过程分为前端和后端，可以实现多种源语言和多种目标语言的灵活组合，消除重复开发的工作量。这种划分使得前端可以专注于处理与语言特性相关的部分，而后端则专注于与特定硬件平台相关的优化和代码生成。这样的架构有助于提高编译器的可维护性、可扩展性以及适应不同应用场景的能力。此外，前端和后端的分离还促进了团队之间的分工合作，提高了开发效率

4、

目前常用的高级程序设计语言包括**C语言、C++、Java、Python、JavaScript**，它们各自的特点如下：

- **C语言**：C语言是一种广泛使用的高级语言，以其高效和灵活著称。它具有功能强大、支持复杂的数据结构、可大量运用指针，以及丰富灵活的操作运算符及数据处理操作符等特点。C语言还具有汇编语言的某些特征，使得程序运行效率高，广泛应用于底层开发。

  类型：编译型。

- **C++**：C++是在C语言基础上发展起来的面向对象编程语言。它具有高性能、泛型编程和模板编程的能力，设计思想是面向对象的，可以编写出高效、安全的代码。C++广泛应用于系统级编程、游戏开发、嵌入式开发、科学计算等领域。

  类型：编译型。

- **Java**：Java是一种跨平台的面向对象编程语言，具有强大的跨平台性能、高效性、可靠性以及安全特性。Java常用于开发Web应用、移动应用、桌面应用、企业级应用等，同时也是Android应用开发的主要语言。

  类型：编译解释型（源代码被编译成字节码，字节码被JVM解释执行）。

- **Python**：Python是一种易于学习和使用的高级编程语言，被广泛用于数据科学、人工智能、Web开发、网络编程、自动化等方面。Python的语法简洁、可读性强，拥有大量的库和框架，可以快速开发各种应用。

  类型：解释型。

- **JavaScript**：JavaScript是一种流行的脚本语言，主要用于Web前端开发，同时也可以作为后端开发语言，如**Node.js**。JavaScript具有互动性、跨浏览器特性、易于学习等特点，同时也有较为严格的标准和规范。

  类型：解释型，现代JavaScript引擎如V8也采用即时编译技术。

补充作业

JavaScript引擎中的JIT（Just-In-Time）编译器是一种特殊的执行模式，用于提高JavaScript代码的执行效率。JIT编译是介于解释执行和静态编译之间的技术。

JavaScript Engine中的JIT编译器

- 工作原理：在执行JavaScript代码时，引擎最初使用解释器直接执行代码。当某部分代码被频繁使用时，JIT编译器将这些热点代码（hot code）编译成机器码，存储在内存中，以便后续执行时直接使用编译后的代码，避免每次都解释执行，从而大幅提高执行效率。
- 优化策略：JIT编译器通常包含多个优化阶段。例如，初级优化可能只进行基本的优化以快速编译，而更高级的优化阶段会进行更深入的代码分析和更复杂的优化技术，如内联展开、死码消除等。
- 回滚机制：如果JIT做出的优化假设在后续执行中不成立（例如，类型预测失败），编译后的代码会被丢弃，系统将退回到解释执行或者重新进行JIT编译。

JIT与编译型语言的区别和联系

- 编译型语言：
  - 编译过程：编译型语言在程序运行之前，源代码会被编译成机器码，这个过程一般只进行一次。这意味着编译过程可能相对较慢，但运行时的效率较高。
  - 与JIT的联系：一些编译型语言（如Java）也采用JIT技术来进一步提升运行时性能，通过在运行时进行额外的优化。
- 解释型语言：

- 解释型语言（如传统的JavaScript）：代码通常在执行时逐行被解释器读取并执行，不经过预编译为机器码的过程。这使得初次执行速度快，但总体性能低于编译型语言。
- 与JIT的联系：JIT技术正是为了弥合解释型语言在执行效率上的不足。通过将热点代码编译成机器码，JIT使得解释型语言在运行时接近编译型语言的执行效率。