编译课程实践任务书

武汉大学计算机学院

2025 年 6 月

目录

1 任务概述

四人为一组，实现 ToyC 语言（见第 2 节）的编译器，将 ToyC 源文件经过词法分析、语法分析、语义分析、代码生成等步骤，编译成能够正确执行的 RISC-V32 汇编文件（扩展名为 s），并完成一份实践报告。你的编译器实现将由在线评测系统进行自动测试并评分。

1.1 实现要求

你可以使用 C++（C++20）、Java（JDK 21）或 OCaml（5.3.0）中的任意一种语言来实现你的编译器。评测时的编译时间限制很宽松，且编译器本身的执行效率不影响评分，所以建议选择你最擅长的语言来完成。鼓励编译器中实现多种优化算法，以提高所生成汇编代码的运行效率。禁止抄袭代码或套用 Clang、LLVM 等现成框架。

1.2 提交与评测

你需要将你的编译器项目上传到希冀平台的 GitLab 仓库中。提交评测时，你需要指定一个分支，评测系统将克隆指定分支的代码到评测环境，并进行构建和评测（项目结构要求和具体评测流程稍后将在另一份文件中说明）。除项目外，每组还需要提交一份实践报告文档和一份能够反映项目组成员工作过程的代码仓库提交记录。

1.3 测试用例与评分标准

所有测试用例的 ToyC 源文件均无语法错误，符合该语言的所有语义约束，且不包含未定义行为。源文件编译成的可执行文件能在有限时间内执行完成。程序的运行结果以主函数的返回值（即程序的退出代码，是 0~255 之间的整数）为准。当生成的代码在规定时间内完成运行，且运行结果与正确结果一致时，测试用例视为通过。

最终评分将综合考虑测试用例的通过率（正确性）、生成代码的运行时间（性能）和实践报告内容。每个测试用例的性能得分将以 gcc -O2 生成的代码运行时间作为基准进行计算，以满分封顶：

$$\text{总得分} = \text{评测得分} \times 80\% + \text{实践报告得分} \times 20\%$$

$$\text{评测得分} = \text{正确性得分} \times 85\% + \text{性能得分} \times 15\%$$

$$\text{正确性得分} = \frac{\text{通过的测试用例数量}}{\text{测试用例总数量}} \times 100$$

$$\text{性能得分} = \frac{\sum_{\text{测试用例}} \min\left\{1, \dfrac{\text{基准时间}}{\text{实际运行时间}}\right\}}{\text{测试用例总数量}} \times 100$$

2 ToyC 语言定义

ToyC 是 C 语言的一个简化的子集，为本课程实践的练习而设计，源文件的扩展名为 `tc`。相较于 C 语言，ToyC 省去了数组、指针、I/O、前向声明、多文件编译等特性，语法也更为简单。ToyC 源文件可以用 C 编译器编译成可执行文件，其运行结果与等同的 C 代码编译成的可执行文件一致。

2.1 ToyC 语言的文法

ToyC 语言的文法如下，其中 `CompUnit` 为开始符号：

编译单元	`CompUnit` → `FuncDef`$^+$
语句	`Stmt` → `Block` | “;” | `Expr` “;” | `ID` “=” `Expr` “;”
	| “int” `ID` “=” `Expr` “;”
	| “if” “(” `Expr` “)” `Stmt` (“else” `Stmt`)?
	| “while” “(” `Expr` “)” `Stmt`
	| “break” “;” | “continue” “;” | “return” “;”
语句块	`Block` → “{” `Stmt`* “}”
函数定义	`FuncDef` → (“int” | “void”) `ID` “(” (`Param` (“,” `Param`)*)? “)” `Block`
形参	`Param` → “int” `ID`
表达式	`Expr` → `LOrExpr`
逻辑或表达式	`LOrExpr` → `LAndExpr` | `LOrExpr` “||” `LAndExpr`
逻辑与表达式	`LAndExpr` → `RelExpr` | `LAndExpr` “&&” `RelExpr`
关系表达式	`RelExpr` → `AddExpr`
	| `RelExpr` (“<” | “>” | “<=” | “>=” |
	“==” | “!=”) `AddExpr`
加减表达式	`AddExpr` → `MulExpr`
	| `AddExpr` (“+” | “-”) `MulExpr`
乘除模表达式	`MulExpr` → `UnaryExpr`
	| `MulExpr` (“*” | “/” | “%”) `UnaryExpr`
一元表达式	`UnaryExpr` → `PrimaryExpr`
	| (“+” | “-” | “!”) `UnaryExpr`
基本表达式	`PrimaryExpr` → `ID` | `NUMBER` | “(” `Expr` “)”
	| `ID` “(” (`Expr` (“,” `Expr`)*)? “)”

2.2 ToyC 语言的终结符特征

标识符 终结符 `ID` 识别符合 C 规范的标识符。其正则表达式为：`[_A-Za-z][_A-Za-z0-9]*`

整数 终结符 NUMBER 识别十进制整数常量。其正则表达式为：`-?(0|[1-9][0-9]*)`

空白字符和注释 忽略空白字符和注释。空白字符包括空格、制表符、换行符等；注释的规范与 C 语言一致：单行注释以 "//" 开始、到最近的换行符前结束，多行注释以 "/*" 开始、以最近的 "*/" 结束）。

2.3 ToyC 语言的语义约束

程序结构 程序必须包含一个名为 `main`、参数列表为空、返回类型为 int 的函数作为入口点。

函数 ToyC 中的函数支持传入若干 int 类型的参数，返回值可以为 int 或 void 类型。函数只能声明在全局作用域中且名称唯一，不能声明在函数体内。函数不能作为值来储存、传递或运算。函数调用必须写在被调函数声明之后（支持函数内部调用自身）。返回类型为 int 的函数必须在每一条可能的执行路径上通过 return 语句返回一个 int 类型的值。返回类型为 void 的函数可以不包含 return 语句，如果包含，则不能有返回值。

语句 ToyC 中的语句包括语句块（由大括号包围的语句序列）、空语句（只有分号）、表达式语句（表达式加分号）、变量赋值、变量声明、if-else 条件分支、while 循环、break、continue 和 return，它们的语法和语义与 C 语言一致：没有返回值的函数调用表达式不能作为 if/while 条件或赋值语句的右值；break 和 continue 只能出现在循环中等。

变量声明 ToyC 支持声明 32 位有符号整数（int）类型的局部变量，每个变量声明语句只声明一个变量，且必须带有初始化表达式。变量的使用必须发生在声明之后。变量的作用域规则与 C 语言一致：变量的生命周期从声明点开始，到所在作用域结束时结束；函数形参在函数体内可见；语句块 `Block` 会创建新的作用域；内层作用域会屏蔽外层作用域的同名变量等。

表达式 ToyC 支持逻辑运算、关系运算、算术运算、括号表达式、变量引用、字面值和函数调用等 C 语言中的常见表达式。运算符的优先级、结合性和计算规则等与 C 语言一致：逻辑与、逻辑或运算符遵循 "短路计算" 规则；非零视为 "真"、零视为 "假"；除数不能为零等。