文章编号：　　中图分类号：G642　　文献标识码：B/A

编译原理实践改革的再次探索：引入开源编译器 LLVM

张昱，桑榆扬

(中国科学技术大学 计算机科学与技术学院，安徽省 合肥市 邮编 230027)

摘　要：针对计算机及相关专业的毕业生在就业过程中暴露出的对编译过程理解不足、动手能力差等问题，以及开源编译器 LLVM 的广泛使用和模块化设计的优势，提出结合 LLVM 的编译原理课程实践的新方案。新方案在 2015 年秋季学期的编译原理课程教学中进行了实施，促进了学生对系统能力的培养,也让学生近距离接触产品级编译器的实现。本文结合具体实施情况，总结了该实践方案的内容、方法、效果和经验教训。

关键词：编译原理；实践；开源编译器；系统能力

随着学术界和工业界对计算机人才的要求不断提高，计算机及相关专业的不少本科毕业生在读研或就业过程中暴露出编码能力弱、对编译过程理解不足、不能灵活运用常用算法和数据结构、不熟悉软件工程的开发流程和相关工具等问题。这些问题的出现很大程度上反映出高校在学科的专业实践（特别是课程实践）教学方面的不足。

“编译原理”作为计算机科学与技术专业的“经典”核心课程，其涉及的原理和技术具有十分普遍的意义，也是培养学生系统能力（系统的认知、分析、开发与应用）和形式化描述能力的一门非常好的课程[1]。然而，随着高等教育的大众化，许多高校对计算机专业是否需要开设“编译”课程出现了疑问，对该课程采取取消或者减少课时、教学偏重灌输书本知识、严重缺乏课程实践以及理论联系实际的讲解等。这一方面不利于培养和提高学生的系统能力，另一方面也难以向由研究生担任助教的高校输送能胜任“编译”课程助教的学生。

笔者多年来从事编译原理课程的教学与改革，在[2]中曾介绍由学生独立完成编译器前端或后端的综合性课程实践方案。这一方案在 2007 年到 2010 年的编译课程中实施，对学生系统能力的培养有很大的促进，但因未拆解成循序渐进的多个任务、要求学生使用多种不熟悉的编程语言和工具（如采用非必修的 Java 语言来开发）等而导致学生上手困难。2012 年起，编译课程实践改成主要用 C/C++语言开发的、以最终构建编译器为目标的多个循序渐进的任务组成。这期间，课程教学面临的主要问题之一是难以找到能胜任的助教。为解决助教问题，2014 年起，笔者在学院支持下尝试选用上年度上过该课程的优秀大四本科生作为助教，初显成效。2015 年，由于助教（即本文的第 2 作者）非常得力，且鉴于开源编译器 LLVM(http://llvm.org/)在学术界和工业界的广泛使用和良好的模块化、文档化等优势，笔者研制了引入 LLVM 的新的课程实践内容，以期让学生近距离接触产品级编译器的实现。本文重点讨论该课程实践方案、实施效果，总结课程实践中的经验教训，供同行参考。

1 课程实践存在的问题与定位

在众多高校中，课程实践教学普遍存在以下问题：

1) 各课程的课程实践各自独立，实践覆盖面窄、综合性不高、难度低、规模小，不太注重对学生工程、质量、团队等意识的培养。

2）课程偏重介绍和考核理论及书本知识，理论与实践脱节，学生感到教学内容抽象、枯燥。实践部分占总评的比重不高，使学生不重视实验，平时对编码能力的训练不足。

3) 教材和实践内容脱离前沿、内容陈旧、跟不上计算机科学与技术的发展，实践中使用的工具和方法在实际工作中已被淘汰或不常用。

4）助教数量不足，对课程内容不够熟悉，检查力度不足，学生提交实验时容易蒙混过关；并且学生由于课业繁重等原因，倾向于互相抄袭。

针对这种现状，笔者早在 10 年前就认为应整体规划各计算机专业课的课程实践，且指出其覆盖的程度依赖于制度的保证、学科机构的资源以及教职人员的利益[2]。就软件类课程而言，其实践的整体目标是学生至少能参与完成一个有一定规模的软件项目的设计与开发，这样的项目应能涉及到对多门课程所学原理的综合运用。在规划课程实验时，应遵循由小到大、循序渐进的原则，注意整体规划实验所涉及的语言、工具和环境，注意培养学生的软件工程意识、质量意识和团队意识，提升学生的计算机系统能力。

落实到具体的课程，需要增加实践在课程总评中所占的比重，改革实践内容；既要提升学生对编程的重视和兴趣，也要科学公平地规定评分标准、减少互相抄袭的可能性。“编译原理”作为本科高年级的软件类课程，建立在学生已有一定编程能力的前提下，旨在提升对编译过程的理解，并通过构建编译器训练学生设计、开发有一定规模的软件项目。

2 编译课程实践改革的历程

基于上面的课程实践定位，2004 年起笔者带领学生调研国外典型编译原理的实验内容。在此基础上，结合自身的教学实践经验，设计了一套以“源语言－抽象语法树(AST)－低级中间表示－汇编代码的内部表示－x86/MIPS 汇编”为主线搭建的、基于组件的编译原理实验体系，并编写了实验教程[3]，研制了配套的实验支持库和课程设计开发包。这套实验体系包含各种循序渐进、规模适度、“综观全局、实现局部”、强调工程质量规范的课程设计。

在 2007 年和 2008 年春季我院编译课程教学中，并没有布置上述实验体系中循序渐进的小实验，而是把支持库和软件包发布给学生，要求学生：用 Java 语言为 SkipOOMiniJOOL 语言（Java 语言的一个小子集）开发编译器的前端（或后端），并自行选择完成后端（或前端）的合作伙伴，组成一个完整的编译器。前端需要完成词法分析、语法分析、语义检查并生成抽象语法树；后端需要由抽象语法树生成 x86 汇编码并进一步用 gcc 汇编并连接得到可执行文件，后端不要求代码优化；前后端接口是抽象语法树，实现前后端的同学相互不得开放源码。2009 年和 2010 年，针对先前两年暴露出的问题[2]，着重加强了课程实践的过程管理，增加了学生提交中间结果的次数和给学生集中答疑的次数，并鼓励学生有技术问题可随时发邮件咨询或在 BBS 上进行讨论。在课程实践内容上，增加了以低级中间表示作为前后端之间接口的选择，增加了以 MIPS 汇编码作为目标代码的选择。在课程设计开发包上，新增一些可用的编译器组件，方便学生在实验初期利用它们构造编译器，并通过了解这些组件的输入和输出等来引导自己的实验。

这套课程实践方案涉及多种编程工具与环境，包括：Java 语言及其开发运行环境，JFlex、CUP、JavaCC 等编译器的生成工具，Ant 编译工具，GCC 以及 SPIM 模拟器等；并且课程实践的开发规模接近工程实际，是一个很好的软件工程实践实例。许多同学认为这个课程实践方案设计得较好，是他们在大学期间所做的最大的实验，在巩固编译原理的知识和提高软件开发能力上有很大作用。如果实践指导力量跟得上，学生的工程质量规范和综合运用知识等能力也会有很大提高。不过，由于一些学生不熟悉 Java 语言和相关工具，也导致他们在开展实验时上手慢，有的甚至有畏难退缩的现象。

为消除学生因编程语言不了解而引起的额外负担，2011 年起，编译实验主要采用 C/C++语言编写。而为了更好地引导学生开展实验，2012 年起，课程实践由多个循序渐进的小实验组成。2012 年是本院编译课程设置的转折点:一方面该课程由三年级下学期前移至三上，以平衡大三课程的负担；另一方面从 2010 级开始，编译课程分成普通和 honor 两个级别，两个班同时并行开课，它们的主要差异体现在课程实验的量和深度上。从 2010 级开始，honor 班（约 30 人）的课程实践除包含多个小实验外，还包含最后的扩展实验。从 2012 年秋季起，笔者从学院申请专门的长时间在线的服务器，为每位 honor 班学生建立只能由学生本人和老师访问的独立的 git 库，学生自行管理维护该库中的目录结构，及时提交其课程实验成果。

由于分级教学，honor 班面临的问题之一是难以找到能胜任的助教。为此，2014 年起，在学院支持下尝试选用上年度上过 honor 课程的优秀大四学生作为助教，初显成效。2015 年，由于助教非常得力，且开源编译器 LLVM 在学术界和工业界广泛使用、有良好的模块化和文档化优势，笔者将 LLVM 引入到课程实践中，以便让学生了解产品级编译器的实现。

3 2015 年课程实践的新方案

2015 年秋 honor 班编译课程的所有实验由 7 个小实验组成的基础实验和最后的扩展实验构成，普通班的实验主要是从基础实验中进行选取。基础实验的最终目标是完成一个 C1 语言的编译器，涵盖了词法分析、语法分析、语义检查、代码生成，并根据课程进度，穿插了阅读 Clang 源码、学习词法

分析器的生成工具 Flex 和语法分析器的生成工具 Bison 的用法、学习 LLVM 中间表示(IR)及其操作接口等任务。扩展实验可以是对之前基础实验的扩展改进，也可以自由选择其他的内容进行探索。

3.1 C1 语言的主要特征

C1 语言是 C99 的子集，它包含了变量和常量声明、变量赋值、while、if-else、无参数无返回值的函数调用、算术运算和比较运算，仅支持 int 一种类型（因此也不支持逻辑运算），不支持 include、多文件编译。通过设计该语言的编译器，可将课程中涉及到的大部分概念带入实践中。

3.2 Kaleidoscope 和 Clang

Kaleidoscope 是 LLVM 网上教程（http://llvm.org/releases/3.6.0/docs/tutorial）提供的 toy 语言，它给出了一个简单的函数式语言的编译器实现，并且增加了命令式语言特性的扩展，有完整的教程讲述如何搭建编译器。它的词法分析部分使用字符串匹配、语法分析用了 LL 文法、后端使用了 LLVM 的接口，也穿插讲了 LLVM 后端的一些概念。

Clang（http://clang.llvm.org/）是目前 LLVM 使用的 C/C++编译器前端。阅读该源码旨在让学生了解一个真实的编译器是什么样的，了解 Clang 如何记录错误、如何处理二元运算时的优先级和结合性、如何构建静态分析的接口等。通过阅读源码，不仅了解一些编译技术，而且可以感受一些软件工程的思想，部分学生在学习后可以应用到 C1 编译器的开发中。

3.3 基础实验的主要任务

基础实验的主要内容详见 https://www.zybuluo.com/clarazhang/note/190166。它包含如下 7 个小实验：

1）**预备阶段**：熟悉实验环境(Linux、LLVM/Clang、GCC、Makefile)以及 C1 语言特征。

2）**词法分析**：学习 Flex 并用它为 C1 语言构造词法分析器，理解 Kaleidoscope 的词法分析过程以及 Clang 词法分析文件。

3)**Kaleidoscope 语法分析**：理解 Kaleidoscope 的 LL 语法分析和 AST，并扩展增加 while。

4）**C1 语言的语法分析**：学习 Bison 并用 Flex 和 Bison 构造 C1 的 LALR 分析器。

5）**生成 C1 程序的 AST**：理解所提供的案例，为 C1 构造能生成 AST 的分析器。

6) **Clang 语法分析和静态检查**：阅读 Clang 源码中指定的源文件，理解其语法分析和静态语义检查的实现机制。学生可以在这个实验中选做检查器等附加实验内容。

7）**生成 LLVM IR**：理解 LLVM 的 IR，为 C1 程序生成 LLVM IR 并利用 LLVM 的工具链得到可执行文件。学生可以在这个实验中对 C1 语言做适量扩展并实现对所做扩展的编译器支持。

每个实验都规定有提交细则和提交的目录结构，引导学生对工程、质量等意识的培养。

引入 Kaleidoscope 教程的阅读，一方面可以让学生了解其 LL 分析器的实现机制，弥补单纯用 Bison 构造 LALR 分析器的不足；另一方面，希望学生阅读完该教程后能基本掌握 LLVM IR 的接口，部分没有涵盖的内容也可以通过上网搜索、阅读文档、讨论交流等形式学习。

针对 Clang 源码阅读，采取由助教先深入阅读理解，找出有代表性的部分并针对其设计题目，然后和主讲教师一起研讨、确定并写出源码导读，指出需要学生阅读的源文件及其中的代码段并回答问题。这样可以使学生不至于在面对大规模代码时无从下手，大大减轻学生的压力，并且可以掠过不少细节性的代码，将精力投入到编译技术的学习上。

3.4 扩展实验的内容选择

扩展实验一般在 11 月底要求学生上报选题，描述拟实现的具体内容。以往的扩展实验（http://staff.ustc.edu.cn/~yuzhang/compiler/eprojlist.html#old 列出了历届选题）是要求学生自拟题目并独立完成，这一方面产生了一些很有创意的成果（如设计数据流语言、用 JS 编写 C 编译器等），另一方面导致部分学生选择了无力完成的题目以至于完成情况较差。2015 年秋季学期，笔者给学生两个选择：扩展 C1 或自拟题目，自拟题目要求事先和教师讨论，教师对选题的工作量、是否有价值等把关；部分选题类似的学生可以合作完成，事实表明这一尝试的效果相比以往要好得多。在 31 位选 honor 课程的学生中，20 人权衡课业负担，理性地选择扩展 C1，剩余学生的自拟题目包括：2 名学生合作开展对数据流语言 StreamIt 的调

研并在多核、多节点机器上进行性能评估和分析；3 名学生合作开展对代码克隆及相似度检测技术的调研和实践；其他学生独立开展宏汇编器的研发、在 LLVM 上通过剥离和重组结构体数组进行静态内存布局的优化、Go 语言调研或 Python 编译器调研等等。

3.5 考核方法

学生的成绩由平时作业（占 10%）、期中和期末的卷面成绩（各占 20%）、基础实验和扩展实验、讨论课表现、最后的逐个人的实验答辩评测（共占 50%）等因素决定。

对于作业仅要求按时交，不考察正确率，笔者希望此举可以减少学生互相抄袭和抄答案的行为。期中和期末考试均采用开卷笔试，但不得使用任何电子设备，每次 1~1.5 小时。采用开卷是希望学生不要死记硬背，而主要依据平时理解的知识来答题。

学生需要将平时的实验提交到自己专有的 Git 库中，由助教批改后在 Git 库中给出批改意见，分数不会告诉给学生。实验批改中主要考察实验文档和程序运行情况，并不花太多时间阅读代码。有的学生对实验做了扩展，如进行语法检查时支持较多特性、给 C1 增加带参函数等都会酌情加分并反映在批改意见中，笔者希望借此激励学生投入更多精力。第 6 次实验主要是阅读 Clang 语法分析和静态检查，其中给出了两个“附加题”：编写 Checker 插件和阅读 Malloc Checker。这两个工作都需要投入较多精力，深入到 Clang 内部研究其原理。因此，若学生完成附加题，则可以加更多的分，实际约一半的学生完成了附加题。

对于最后的扩展实验，将学生分为 4 组，每组约 8 人参与答辩。评委由所有学生、助教和主讲老师组成，任何人都可以对演讲者提问，但是整个过程由教师主导，提问主要集中在对代码的功能、正确性、知识的掌握程度等方面。所有评委根据学生演讲和回答的情况给出组内排名，最终的成绩由分组的综合排名和教师评分确定。

3.6 互动

2104 年学生和教师间的交流使用 google group，但是由于需要翻墙所以学生参与度不高。2015 年秋季学期主要通过邮件和 QQ 群交流。学生如果有疑问或意见，大多通过邮件向老师反映，老师会将有建设性的内容群发给所有学生。此外，关于实验的疑惑大多在 QQ 群内交流，学生在此处的参与度较高，许多问题在内部就解决了；老师也会在 QQ 群引导解决部分棘手的问题。

4 实施效果与经验教训

4.1 实施效果

笔者根据 31 位学生的 git 库中的各次提交信息统计了提交日期(图 1)与提交时间(图 2)的分布。

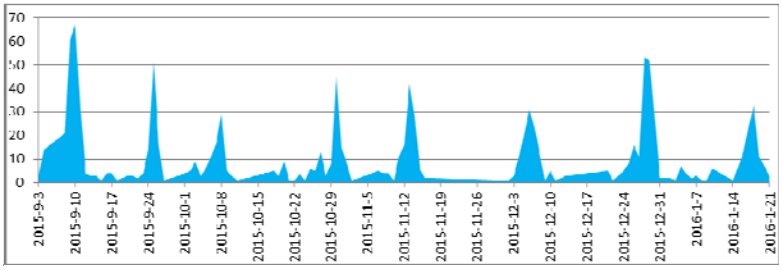

图 1 学生 git 库提交的日期分布

从图 1 可以清楚地看到有 8 个明显的高峰，它们分别对应了 7 次平时实验与扩展实验的提交截止日期，分别为 9.12、9.26、10.9、10.31、11.15、12.7、12.29、1.18。这表明同学们依然偏向于拖延到最后一刻提交；并且更喜欢在前一天晚上集中突击。由于截止时间定在上午 7 点前，因此从图 2 可见，16 时至凌晨 0 时以及 6 时的提交量较高，1 时到 5 时的提交量显著降低，这是因为学校本科生宿舍晚上断电，只有少数学生可以在实验室通宵写代码。

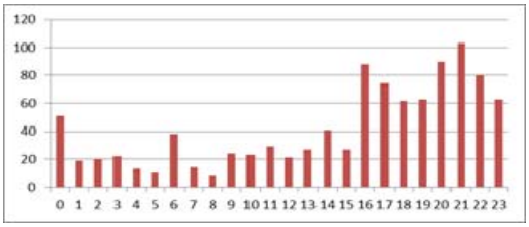

图 2 学生 git 库提交的时间分布

表 1　学生提交的平均代码量统计

	C++	C	头文件	Lex	Yacc	Makefile
文件数	21.97	23.8	15.9	3.27	3.57	6.1
代码行数	9021.57	3852.53	1373.6	275.8	2014.93	175.27

表 1 是利用代码统计工具 cloc 得到的学生提交的平均代码量，可以看到 C++占了绝大

部分，C 和 Yacc 也有不小的比例，这是因为需要用 C++编写 AST 节点的构造、遍历、处理的函数，产生可执行代码时也要用 C++调用 LLVM 的后端；C 通常作为测试程序出现，也有学生喜欢写 C 代码；而 Yacc 文法描述代码之所以这么多是因为需要在产生式后编写生成 AST 节点的操作，而部分学生将一些后续处理的代码也放了进去，这是笔者不提倡的。

4.2 经验教训

1）**和 LLVM 的结合**：LLVM 是一个前景极佳的项目，它的编译速度可以碾压 gcc，同时相对于 gcc 可怜的文档也更适合构建编译器。因此笔者希望学生熟悉 LLVM，即使将来不从事编译相关的工作，也可以通过课程实践学到实用的知识。在第 1 个基础实验中就要求学生了解 LLVM 的指令和含义，此后也加入了阅读 Clang 源码的作业，借此学生可以了解软件开发中的代码规范、为了避免 bug 而产生的设计技巧等，还可将所学应用到实验中；而最终学生还要用 LLVM 的后端接口生成 IR，LLVM 的教程涉及了部分概念，但是也遗漏了部分信息，因此笔者鼓励学生上网查找、阅读文档，并且笔者也提供了涵盖大部分接口的样例代码。

2）**扩展实验**：扩展实验分为两类：改进 C1 编译器和自拟题目，既允许时间不足的学生完成作业，也能让学有余力的学生有机会接触到更前沿的知识。由教师主导，学生参与评分的扩展实验答辩方式，可以提高学生参与的热情、增加给分的公平性和公开性。

3）**赶截止日期**：编译原理作为本院学生在本科期间实验量最大的课程，可以帮助学生提升编程能力、提高对编译链的理解，但由于工作量大也给学生带来了较大压力。许多学生偏好在实验最终提交日期前临时突击，导致投入时间不够、完成情况不好甚至无法完成。这是一个无法避免的问题，因此笔者尽量将工作任务细分，给每个部分规定截止时间。

4）**工具**：本院的传统是鼓励学生研究理论而缺少对实践的强调，许多学生到了大三依然没用过 Linux。本实验要求学生在 Linix 环境下编程、使用 Git 提交作业、用 make 辅助编译、用 shell 编写批量测试程序，此外还要学习 Lex、Bison、Clang、DOT 等工具，在编译、调试过程中整条编译链的工具都需要掌握。这些工具是做编译实验必须的，学生需要花不少时间学习它们，而不能将精力集中到编译技术上来。笔者认为学院应该重视学生基本编程能力的培养，在低年级课程中增加对常用工具的介绍和使用；在本课程中可提供简单的教程帮助学生迅速上手。

5）**使用 Git**：使用 Git 的好处是可以方便地确定学生的提交时间、助教可以随时提取学生作业，不需要安排固定时间检查；而且学生在提交后可以方便地修改，只要记得经常同步，即使本地版本丢失也可从服务器上获得版本继续开展实验。但是，实际使用时发现有的学生的提交次数不多，一部分是因为赶在截止日期前完成，另一部分是喜欢写完所有的代码之后一次性提交，而且写的日志也无法反映出足够的信息，如使用“P5”、“P5.1”之类的日志信息等。这可能是由于每个人负责一个 Git 项目，不存在合作，所以学生觉得没有必要写详细的日志；而且学生缺乏编程项目的实践，故也没有多次提交的意识。笔者考虑今后加入学生互评的机制，充分利用 Git 的协作性，让学生通过阅读他人代码、介绍别人的批改意见来相互促进、消化知识。

6）**提供样例代码**：助教在实验中数次给出过样例代码，如在构建 C1 编译器初期给出过仅包含赋值、算数运算的语言的编译器代码，包含了 Lex、Bison、Makefile、类的组织等。不少学生的代码就是基于此，在一定程度上减轻了学习、调试的难度。构建 C1 编译器末期，许多学生反映 LLVM 的接口太难懂，因此助教又给出了一份包含大部分接口用法的样例代码，并提供了根据一份 C 源码还原出它的 IR 用到的 LLVM 接口的方法。提供样例代码，可以减轻学生负担，但如果过于详细由会导致学生无事可干，因此要注意点到为止。

7）**学生的编码质量**：基础实验往往存在有依赖关系，即后一个实验可能在前一个的结果上开展。这种依赖关系也带来了弊病：如果学生的某次实验完成得不好，会影响此后依赖它的其他实验。事实上，部分学生在期末向笔者反映自己的代码构建得过于杂乱，导致后期调试和修改起来非常困难。针对该问题，笔者也做了些预防措施，如在前期就提供一个简单语言的编译器的样例代码；但到了后期要添加的特性变多，不同实现导致工作量差别较大：如有的学生自定义了一个类处理符号表，从而在遍历 AST 时方便操作；而有的直接在每个需要操作符号表的地方手工处理，导致工作量翻了数倍。

8）**小班教学**：总体上这种 30 人左右的小班教学优于大班教学，主要体现在：课堂教学容易互动、日常实验容易管理、最终答辩容易实施这三个方面。在小班课堂中，学生比较活跃，课间和课后问问题比较踊跃；老师可以及时询问了解学生的进度和存在的问题，调整实验提交的截止时间，也能在课间和学生个别交流其学习存在的问题及原因。

5 结语

本次课程实践基于以往的经验及教训，基于 LLVM 重新设计了部分实验内容，并修改了扩展实验的选题与考核机制等，有效地帮助学生更好地理解编译知识；其中也暴露出实验设计上的一些问题，有助于我们将来进一步的改进。

参考文献：

[1] 蒋宗礼. “编译原理”课程与专业能力培养[J]. 计算机教育,2009(21):4-6.
[2] 张昱，陈意云. 编译原理课程实践改革探索[J].计算机教育，2008(8)：24-26.
[3] 张昱，陈意云. 编译原理实验教程[M].北京：高等教育出版社，2009.5.

基金项目：安徽省 2013 年省级精品资源共享课程《编译原理和技术》（皖教高[2013]11 号，项目编号 2013gxk002）。
作者简介：张昱，女，副教授，研究方向为程序设计语言理论与实现技术、面向多核的可靠高效并行系统软件的构建与评估、软件安全等，yuzhang@ustc.edu.cn。桑榆扬，男，2012 级本科生，2015 年秋季编译课程的助教，sangyy@mail.ustc.edu.cn。

联系方式：张昱，13866131689，yuzhang@ustc.edu.cn