编译原理实验：Lab3

指导老师：徐辉

2021 年秋季学期

1 实验介绍

Lab3 的目标是在前两个 Lab 的基础上，完成中间代码生成以及可执行的机器码的生成，让我们自己实现的编译器编译出来的文件可以真正地跑起来。这两部分工作中，由于机器码生成的部分较为复杂，而且涉及较多的指令集知识，因此这部分在框架中直接调用了 LLVM 自带的 API 完成了生成，因此对于 Lab3 来说实际上只需要关注中间代码的生成。

1.1 中间代码

中间代码（IR，intermediate representation），也叫中间表示，是一种容易翻译成目标程序和源程序的等效内部表示代码。在编译过程中，可以大致分为前端和后端两部分，其中，前端部分最终得到的结果就是中间代码，而后端只需要关注中间代码，在其基础上进行处理优化，便能得到最后的目标程序。狭义的 LLVM 便是这么一个后端框架，因为 LLVM 有自己定义的一套中间代码表示的语法，只需要通过实现前端程序，将目标语言的源码编译生成 LLVM 的中间代码，便可以直接调用 LLVM 后端框架进行优化得到目标文件 (比如针对 c 实现的 clang 便是这么一个前端框架)。

本实验所需要完成的就是针对 lab2 得到的抽象语法树，生成对应的 LLVM 中间代码，再经由框架提供好的后端部分，由中间层代码生成目标文件，结合 c 语言的 runtime(运行环境)，正确地执行我们编译器编译出来的文件。所以可以说此次 Lab 最后完成的结果是一个真正可以用的 mini 编译器。

2 代码逻辑与 API 介绍

在生成中间层代码部分使用了很多 LLVM 框架中的 API，这些 API 非常优雅地简化了我们的实现过程，这里简单对这些 API 和 Lab3 的框架逻辑以及一些可能会用到的其他 API 进行说明。

2.1 关键变量介绍

在 Code Generation 部分有几个关键变量要进行说明, 这些变量定义在 Code Generation 部分的头部，为全局变量：

- *TheContext*: LLVMContext 类型，为 LLVM 上下文的全局变量，它不透明地储存了 LLVM engine 的很多相关内容，比如常量以及类型表等。

- *TheModule*: Module 类型，包含了函数，全局变量，符号表入口等信息，可以说是储存了 LLVM 模块中所有相关信息，我们整个 IR 生成的代码都是在其中储存信息，最后通过 TheModule 可以得到目标文件。
- *Builder*: IRBuilder 类型，用于生成中间代码指令并储存于 Module 中，所有中间代码的生成都是源自这个类型。
- *NamedValues*: map 类型，用于储存函数中的变量名 (比如参数中的变量必须储存起来，你函数内部使用才能知道这个变量是不是声明了)。其中，变量的类型是 Value 类型，该类型表示的就是 LLVM 中间代码中变量的类型。
- *FunctionProtos*: map 类型，储存函数定义内容，在 Call 操作的时候使用。

2.2 代码逻辑

首先是 Main Drive 和 Top Level 这一部分主要是编译器的入口，框架都已提供完整不需要进行额外修改。Main Driver 对上一 part 中提到的关键变量进行了初始化，之后转入 Top Level 部分进行词法分析语法分析以及中间代码的生成工作，在最后通过调用 LLVM 框架中的 API，对已经生成好中间代码的 Module 生成目标文件并输出。而 Top Level 部分中，前两个 Lab 中的工作保持不变，额外增加的是在得到 AST 之后，对 AST 结构体进行 codegen() 操作 (该函数已在每个结构体中定义好)。

而 Code Generation 部分则需要对之前 AST 定义的结构体中的 codegen() 函数进行实现。这一 part 已经实现或是部分实现了一些函数以供参考，同时还提供了一些工具函数，比如 getFunction() 用来得到所需要的函数 (这些都可以自由发挥进行修改)。

2.3 框架 API

在 Code Generation 部分有几个可能用到的框架 API 在这里进行说明解释，方便同学们使用理解：

2.3.1 IRBuilder 成员函数

- CreateAdd()：整数相加
- CreateSub()：整数相减
- CreateMul()：整数相乘
- CreateFAdd()：浮点数相加
- CreateFSub()：浮点数相减
- CreateFMul()：浮点数相乘
- CreateICmpULT(): 整数的 < 比较（左值比右值）
- CreateUIToFP(): 整数转浮点数
- CreateFPToUI(): 浮点数转整数
- CreateCall(): 创建函数调用语句
- CreateRet()：创建函数返回值

...

以上为创建中间代码的函数，在框架中，比如有两个 int 值 A、B，想要加法运算得到名为 temp 的变量，可以书写为：

$Builder->CreateAdd(A, B, "temp")$;

返回类型为 Value，即这个运算得到的值，其中中间带 F 的为浮点数运算，也就是 A,B 都必须是浮点类型的 Value(理解为 Value 的一种子类型)

2.3.2 获得类型值的函数

Value 类型有个成员函数为 $getType()$, 可以返回该 Value 的表示的中间代码的数据类型，这个类型值在 LLVM 框架中由 Type 类进行表示，该类有很多成员函数供我们用来判断类型，比如 $isDoubleTy()$ 用于判断 Value 表示的中间代码类型是否是浮点数类型，返回 bool 值。同时，如果想获取对应类型的 Type 值也可以调对应的 API，比如 $Type::getDoubleTy(*TheContext)$ 就可以获得一个 Double 类型的 Type 值 (某些 Create 操作需要)

2.3.3 Example

为了便于理解与书写代码，以下举一个例子。

- 现有有两个 Value 变量：Value *L，*R; 我们要计算两个之和。
- 首先,我们调用 $L->getType()->isDoubleTy()$, 以及 $R->getType()->isDoubleTy()$, 得到的 L 的调用返回结果为 true，R 的为 false，说明 L 是 double 类型的 value，而 R 不是，是 int 类型的。
- 整数加浮点数，结果为浮点数类型，而对应操作应该是两个浮点数相加，因此我们需要将 R 转化为浮点数。可以用 Builder 这么生成中间代码：

```
1 R = Builder->CreateUIToFP(R, Type::getDoubleTy(*TheContext), "tmp"
     );
2 Result = Builder->CreateFAdd(L, R, "addtmp");
```

值得一提的是，类型的一致在中间代码的生成过程中十分重要，生成过程中的类型检查确认将是一个不小的工作。

本部分已经列举了大部分 lab 所需要的 api，如果仍有不清楚的地方，通过课程网站上的引用链接，查找 LLVM 的官方文档会给你更为精确的答案。

3 实验测试

lab3 实验有两个测试用例，首先是将前两个 Lab 内容补充至代码后，不做任何修改的情况下，使用压缩包中的 *input_example.data* 以及提供好的 runtime 进行测试 (主要测试的是环境), 在配置好环境后，在当前目录的命令行下输入如下指令，即可输出测试结果。

```
1 >> clang++ -g -O3 lab3.cpp `llvm-config-9 --cxxflags --ldflags --
     system-libs --libs all` -std=c++17 -o lab3
```

注意，该操作编译时间较长，等待一两分钟是正常现象 (快的话说明电脑好)，同时第一步的这个指令是针对 LLVM 版本为 9.0 的指令，如果版本高于 9.0 不需要在 config 后面加-9，直接如下：

```
(>> clang++ -g -O3 lab3.cpp `llvm-config --cxxflags --ldflags --system
   -libs --libs all` -std=c++17 -o lab3)

>> ./lab3 input\_empty.data
```

其中第一条命令是是编译 lab3，第二条指令是编译文件得到 output.o 目标文件，如果能正确得到目标文件，说明环境正确。当程序实现完成之后，可以把上述指令中的 input_empty.data 换成 input_example.data，得到一个真正可执行的函数。之后使用以下指令：

```
>> clang++ main.cpp output.o -o main
>> ./main
```

这里的第一条指令目标文件一个执行的 runtime。因为我们编译完后，得到的是定义一些函数的目标文件，需要在某个地方被调用执行。一个完善的程序语言书写的程序在开始执行前一般会有一系列复杂的针对运行环境的操作。

这边我们就直接借助 c++ 运行的 runtime，让 cpp 文件内直接调用我们定义的函数来得到结果，因此第三条指令就是对.cpp 文件以及我们编译好的函数库一起编译得到最后的可执行文件，最后一条就是运行可执行文件，检查我们的输出结果。可以与 output_example.data 文件中的内容进行对比。

4 实验提交

实验完成之后，将实验代码在截止日期之前上传至 elearning，助教将根据代码完成质量以及测试用例通过情况进行打分。

提交时只需要提交补充完整可编译运行的代码文件，将代码压缩到以学号命名的压缩包中 (zip 格式)，提交压缩包文件。代码的文件名保持原样，为 lab3.cpp。