**电子科技大学 信息与软件工程 学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**组队排序1 学生姓名：李艳超 学 号：2022090902007**

**指导教师：肖堃**

**实验地点：信软楼 实验时间：2023.11.9**

**一、实验室名称：信软 303**

**二、实验项目名称：LR语法分析**

**三、实验学时：4学时**

**四、实验目的、原理、内容及步骤：**

**实验任务：**

查看指令手册LLVMRef.mht

用clang学习

编写简单的 C语言程序 test.c

用 clang -emit-llvm -S ./test.c 生成该文件对应的llvm指令

学习示例程序

理解代码生成过程

在实验3及示例程序的基础上完成

算术表达式、逻辑表达式

赋值语句、条件语句、循环语句

对应的代码生成，函数命名分别为：

genArithmeticExpr, genLogicExpr

genAssignStmt, genIfStmt, genWhileStmt

函数参数都为： (past node, char\* result)

node为相关类型的结点，

result 为用来保存三地址代码/LLVM虚拟指令的存储空间每行只放一条指令

**实验安排要求：**

尽量通过全部测试用例

需完整总结从词法分析到代码生成各步骤的

技术要点及相关设计，并体现在实验报告中

提交方式

icoding 平台提交，包括实验4全部代码及实验报告

提交截止日期

全年级统一，以icoding平台为准

**参考资料：**

官方资料https://llvm.org/docs/ProgrammersManual.html

版本选择：llvm 3.9.0 有以下

源码编译后，建议调试 opt 程序

观察opt程序如何读取 .ll 文件并解析各条指令的

学习llvm指令中的相关数据结构

**五、实验运行结果：**

**六、实验结论与总结：**

**老师对不起，真的做不出来了，大概梳理一下文件及函数：**

**Ast.c:**

**get\_regs:**

通过接收一个字符指针 name 作为参数来获取寄存器信息。函数首先通过 hash 函数获取 name 的哈希索引，然后在一个名为哈希表中查找与 name 相匹配的条目。通过遍历哈希表的相应链表，如果找到一个条目其名称与 name 相同，则返回该条目的寄存器信息。如果在哈希表中没有找到匹配的条目，函数返回 0.函数用于在编译过程中管理和分配寄存器，将源程序转换为LLVM虚拟指令的环节。

**EstablishTokenTabl：**

遍历给定的抽象语法树（AST）节点 node，并根据不同节点类型执行特定操作。函数的主要功能是处理变量声明（VAR\_DECL）节点，根据其数据类型（整型或浮点型）和属性（如常量或数组），在哈希表中插入相应的条目。

注意：在处理整型和浮点型变量声明时，函数需要考虑是否存在直接的数值字面量（INTEGER\_LITERAL 或 FLOATING\_LITERAL）或初始化列表（INIT\_LIST\_EXPR），并据此在哈希表中插入或更新条目。此外，函数还应该检查变量是否被标记为常量（if\_const），如果是，则相应地更新哈希表条目。

在代码的开始部分，设置一个名为 llvmS 的数组，其中包含了LLVM虚拟指令字符串，遍历AST时生成对应的LLVM代码。

**get\_hash：**

获取给定名称的 phash\_table 条目。首先为提供的 name 计算一个哈希索引，然后在哈希表（预计是 My\_hash\_table）中搜索与 name 匹配的条目。搜索通过遍历哈希索引处的链表进行，直到找到匹配的条目或到达列表末尾。如果找到匹配的条目，就返回它；否则，函数返回 NULL。这个函数通常用于编译器中快速查找标识符的符号表条目，如变量名、函数名等。

**newAstNode：**

为新的AST节点分配内存并进行初始化。使用 malloc 来分配等同于 ast 结构体大小的内存，并检查内存分配是否成功。如果内存分配失败，则打印错误信息并退出程序。为节点分配的内存随后使用 memset 设置为零，确保节点中的所有字段都初始化为默认值。函数对于在编译器的解析阶段构建AST至关重要，在这个阶段，每个节点代表源语言中的一个构造（如表达式、语句等）。

**showAst：**

遍历并打印抽象语法树（AST）的节点。根据节点类型和嵌套层次，以递归的方式展示每个节点的信息。函数支持根据嵌套层级打印不同的缩进，并能够处理不同类型的节点，如整数或浮点字面量。代码中包含了用于展示或生成与AST节点相关的LLVM虚拟指令的部分。

**get\_id 和 get\_stype等函数：**

以get\_id为例：接受一个字符指针 id 作为参数，并返回一个新的字符串，该字符串是 id 的副本。函数首先计算 id 字符串的长度，然后使用 malloc 分配足够的内存来存储该字符串（包括结尾的空字符 \0）。使用 memcpy 将原始字符串复制到新分配的内存中，并返回这个新字符串的指针

**Generatellvm.c:**

**ProcessWhileStmt：**

处理抽象语法树（AST）中的 while 语句节点。函数首先处理条件语句的左右部分，然后根据逻辑运算符的类型（OR 或 AND）生成相应的分支指令。这些分支指令控制程序流根据条件判断结果跳转到不同的标签。该函数是编译器代码生成阶段的一部分，负责将高级语言中的 while 语句转换为底层机器代码或中间表示形式，如LLVM IR，从而实现复杂的逻辑控制流程。

**ProcessLogic：**

处理抽象语法树（AST）中的逻辑表达式节点。检查节点是否为空，然后根据节点的左右子节点类型（整数字面量或其他）来获取或计算它们的值。对于非字面量节点，函数使用 get\_regs 获取相关寄存器的编号，并生成相应的加载指令。函数是编译器代码生成阶段的一部分，负责将逻辑表达式转换为低级代码或中间表示，如LLVM IR，涉及寄存器的管理和指令的生成。

**接下来的几个函数异曲同工，处理if 表达式等，下面分析其他的主要函数**

**Process：**

函数负责将高级语言的函数声明和函数体转换为低级代码或中间表示，如LLVM IR，涉及寄存器的管理、函数声明的解析和函数体的处理。

先处理函数声明（FUNCTION\_DECL）节点。首先检查并排除特定类型的变量声明（VAR\_DECL）节点，然后针对函数声明，重置寄存器计数器并根据函数返回类型（整型、浮点型或无返回值）打印出相应的LLVM函数定义。接着，函数处理函数参数，将它们插入哈希表，并为每个参数生成相应的寄存器和存储指令。此外，它特别处理 main 函数，为其分配和初始化寄存器。最后，函数递归地处理函数体中的各个语句，包括复合语句（COMPOUND\_STMT）和其他可能的语句或表达式节点。函数负责将高级语言的函数声明和函数体转换为低级代码或中间表示，如LLVM IR，涉及寄存器的管理、函数声明的解析和函数体的处理。

**Irlexl.l**

将源代码文本分解成一系列词法单元（如关键字、标识符、数值、运算符等）。通过一系列规则识别注释、空白字符、数字（包括十六进制和浮点数）、运算符和关键字，并对这些元素执行相应的操作，如转换类型、分配内存或计数

Main.c:

调用 yyparse 函数进行语法分析，构建抽象语法树（AST），然后遍历AST来处理全局声明和其他程序结构。具体来说首先从 getNode 获取AST的根节点，使用 showAst 函数打印AST，接着通过 EstablishTokenTable 和 ProcessGlobalDecl 函数处理全局变量和声明，最后遍历AST的右子树来递归处理每个节点

总结

**语法分析和AST构建：**

实验的第一步涉及使用词法分析器和语法分析器来解析源代码。

词法分析器将源代码分解为一系列基本的词法单元，例如关键字、标识符、数值和运算符。

语法分析器根据这些词法单元和预定义的语法规则构建出AST。AST是源代码逻辑和结构的树形表示，其中每个节点代表了源代码中的一个构造，如表达式、语句或函数定义。

**AST遍历与节点处理：**

在AST构建完成后，实验接着要求遍历这棵树。

遍历过程中，根据节点类型（如变量声明、if语句、while循环、函数声明等）来执行特定的操作。

每种类型的节点可能需要不同的处理方式，这可能包括计算表达式的值、确定控制流结构或处理函数调用。

**代码生成和LLVM虚拟指令翻译：**

实验的核心部分是将AST中的每个节点转换为LLVM虚拟指令。

对于不同类型的节点，需要开发相应的翻译函数。例如，一个if语句节点的翻译函数将生成条件分支的LLVM指令，而while循环节点的翻译函数则生成循环控制的指令。

生成的LLVM虚拟指令构成了程序的中间代码表示，这是一种介于高级源代码和低级机器码之间的表现形式。

编译过程从词法分析开始，通过将源代码文本转换为词法单元，接着使用递归下降或LR语法分析来解析这些单元并构建抽象语法树（AST）。递归下降解析适合于简单的语法，通过为每个语法规则编写函数进行处理，而LR分析能更有效地处理复杂语法，使用状态机和分析表来管理符号和规则。最后，在代码生成阶段，AST或中间表示被转换为目标代码，如汇编指令或LLVM中间表示，这一过程涉及寄存器分配、指令选择和高级结构的转换逻辑。整个编译流程确保了从源代码到执行代码的正确且高效转换。

**报告评分：**

**指导教师签字：**