本科实验报告

课程名称 编译原理

姓名

学院

系

专业

学号

指导教师

2021 年 6 月 11 日

目录

序言

概述

paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1 paragraph 1

paragraph 2

编译环境

- Prebuilt environment: Windows Sub Linux Ubuntu 20.04
- Lexer: flex 2.6.4
- Parser: bison 3.5.1
- Framework: LLVM 12.0.0

文件结构

```
Root
|  build.sh  //编译compiler
|  clean.sh  //清理产生的compiler及中间文件
|  run.sh  //运行compiler并将产生的LLVM IR编译为可执行文件
|
├─bin
|  └─ main  //预编译的compiler
├─src
|  |  main.cc  //入口
|  |  Parser.cc  //bison产生的中间文件
|  |  Parser.hh  //bison产生的中间文件
|  |  Scanner.cc  //flex产生的中间文件
|  |
|  ├─ast
|  |      ast.hh  //include-all header
|  |      ast_base.cc
|  |      ast_base.hh //基类node
|  |      ast_const.cc
|  |      ast_const.hh  //常量node，包含各种常量从token到llvm::Value的转换
|  |      ast_expr.cc
|  |      ast_expr.hh  //表达式node，包含表达式中会出现的所有token的解析
|  |      ast_function.cc
|  |      ast_function.hh  //函数node，包含函数的声明与定义
|  |      ast_routine.cc
```

```
│   │       ast_routine.hh  //例程node，包含routine和subroutine的实现
│   │       ast_stmt.cc
│   │       ast_stmt.hh  //语句node，包含赋值、流程控制、过程调用等
│   │       ast_type.cc
│   │       ast_type.hh  //类型node，包含类型的解析和从token到llvm::Type的转换
│   │       ast_variable.cc
│   │       ast_variable.hh  //变量node，包含变量的定义与内存分配
│   │
│   ├─irgen
│   │       generator.cc
│   │       generator.hh  //AST的操作者，实现AST的生成、AST的输出、IR的生成
│   │       table.cc
│   │       table.hh  //符号表的定义
│   │
│   ├─lexer
│   │       Scanner.l  //词法分析器
│   │
│   ├─logger
│   │       logger.cc
│   │       logger.hh  //输出log的工具
│   │
│   └─parser
│           Parser.yy  //语法分析器
│
└─test
        test1.spl  //基本语句的测试
        test2.spl  //给定test
        test2ans.cc  //给定test的c++转写，用于测试结果
        test3.spl  //IO测试
        test4.spl  //给定test
        test4ans.cc
        test5.spl  //数组与记录的使用，函数引用传递的测试
        test6.spl  //给定test
        test6ans.cc
```

分工说明

总览

Compiler Component	**Status**	**Details**
Scanning	Flex	
Parsing	Bison	
Semantic Analysis	AST	
Intermediate Code Generation	LLVM IR	LLVM IRBuilder
Intermediate Code Optimization	LLVM C++ API	LLVM Pass Manager
Target Code Generation	LLVM back-end	Clang++
Target Code Optimization	LLVM back-end	Clang++
Other Optimization	implemented	
Runtime Environment	implemented	全局栈环境
Symbol Table	implemented	

1 词法分析

1.1 正则表达式

1.2 实现原理和方法

2 语法分析

2.1 上下文无关语法

2.2 实现原理和方法

3 语义分析

3.1 实现方法

4 优化考虑

4.1 xxx 阶段优化

5 代码生成

5.1 assign

```
int main()
{
    // comment
    return 0;
}
```

6 测试案例

6.1 单句测试

6.2 组合测试