华中科技大学

编译原理实验报告

专　　业：　计算机科学与技术

班　　级：　CS2008

学　　号：　U202015533

姓　　名：　徐瑞达

电　　话：　17837353795

邮　　箱：　2014027378@qq.com

独创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签名：

日期：2023 年 6 月 15 日

综合成绩	
教师签名	

目 录

1 编译工具链的使用

1.1 实验任务

(1) 编译工具链的使用；

(2) Sysy 语言及运行时库；

(3) 目标平台 arm 的汇编语言；

(4) 目标平台 riscv64 的汇编语言；

以上任务中(1)(2)为必做任务，(3)(4)中任选一个完成即可。

1.2 实验实现

1. 编译工具链使用

在该部分实验，学习使用了 GCC、CLANG、交叉编译器、make 等编译工具。在前三关中，根据实验提示，编写命令行即可。在编写 makefile 文件时，要注意严格的缩进，同时使用 makefile 的自动推导特性，能够大大简化 makefile 文件的编写。

2. Sysy 语言及运行时库

Sysy 语言中，没有 for 语句，也没有--、++、+=、-=等运算符，因此需要在 C 语言代码的基础上进行修改。同时，注意运行时库 getint()用来读取一个整型变量、putint()用来打印一个整型变量、putch()用来打印一个字符（参数为 ASCII 码）。

3. arm 汇编

在本关卡中，通过编写冒泡排序，了解并掌握 arm 汇编语言。在编写时，需要注意参数如何传递（数组 arr 的首地址保存在寄存器 r0 中，数组元素的个数 n 保存在寄存器 r1 中），返回值如何传递（返回值为 0，由 r0 传递），立即数需要加前缀#，指令、伪指令、伪操作、寄存器名等不可以大小写混用，在进入 bubblesort 函数时保存现场（push {r0,r1,r2,r3,r4,r5,r6,r7,r8}），在结束 bubblesort 函数前恢复现场（pop {r0,r1,r2,r3,r4,r5,r6,r7,r8}）。在熟悉 arm 的指令集和寄存器结构后，即

可正确编写代码。代码如下所示：

```
bubblesort:
        push {r0,r1,r2,r3,r4,r5,r6,r7,r8}
        mov r2,#0
        sub r1,r1,1
loop1:
        mov r3,#0
        mov r8,r0
        sub r7,r1,r2
loop2:
        add r4,r3,1
        ldr r5,[r8]
        ldr r6,[r8,#4]
        cmp r5,r6
        ble addj
        str r6,[r8]
        str r5,[r8,#4]
addj:
        add r3,r3,#1
        add r8,r8,#4
        cmp r7,r3
        bgt loop2
        add r2,r2,#1
        cmp r1,r2
        bgt loop1
out:
        pop {r0,r1,r2,r3,r4,r5,r6,r7,r8}
        mov r0,#0
       bx      lr
```

4. riscv64 汇编

在本关卡中，通过编写冒泡排序，了解并掌握 riscv64 汇编语言。riscv64 编程逻辑与 arm 类似，只需注意指令集、寄存器结构即可。代码如下所示：

```
bubblesort:
            addi t0,a1,-1
loop1:
            andi t1,t1,0
            ori t2,a0,0
loop2:
            lw t3,0(t2)
            lw t4,4(t2)
            ble t3,t4,addj
            sw t3,4(t2)
            sw t4,0(t2)
addj:
            addi t1,t1,1
```

```
addi t2,t2,4
blt t1,t0,loop2
addi t0,t0,-1
blt zero,t0,loop1
```

2 词法分析

2.1 实验任务

分别在给出的语法分析器框架的基础上，实现一个 Sysy 语言的语法分析器：

(1) 基于 flex 的 Sysy 词法分析器(C 语言实现)

(2) 基于 flex 的 Sysy 词法分析器(C++实现)

(3) 基于 antlr4 的 Sysy 词法分析器(C++实现)

以上任务任选一个完成即可。

2.2 词法分析器的实现

本关卡选用基于 flex 的 Sysy 词法分析器(C 语言实现)。flex 源文件分为辅助定义、正则式、用户子程序三部分。按照题目要求，只需在正则式部分填写模式和动作即可。

(1) 标识符 ID

标识符应以大小写字母和下划线开头，并接以若干个大小写字母、下划线和数字。其对应语义动作中，打印单词和类型并返回类型 ID 即可。具体代码如下：

```
[a-z_A-Z][a-z_A-Z0-9]* {printf("%s : ID\n", yytext); return ID; }
```

(2) int 型字面量 INT_LIT

int 型字面量包含十进制、八进制、十六进制三种，不同进制的正则式之间以|分隔。十进制字面量以数字 1-9 开头，接以数字 0-9；八进制字面量以 0 开头，接以数字 0-7；十六进制字面量以 0x 或 0X 开头，接以十六进制字符（注意是正闭包）。其对应语义动作中，打印单词和类型并返回类型 INT_LIT 即可。具体代码如下：

```
[1-9][0-9]*|0[0-7]*|0[xX][0-9a-fA-F]+ {printf("%s : INT_LIT\n", yytext); return INT_LIT; }
```

(3) float 型字面量 FLOAT_LIT

float 型字面量包含大于 1 的浮点数、小于 1 的浮点数和科学计数法三种，不同表示法之间以|分隔。具体代码如下：

```
[0-9]+\.[0-9]+f|[0-9]+[\.]*[0-9]*[Ee][+-][0-9]+f    {printf("%s : FLOAT_LIT\n", yytext); return FLOAT_LIT; }
```

(4) 词法错误

当变量以数字开头、十进制数字含有前导 0 时即发生词法错误。

```
[0-9][a-z_A-Z][a-z_A-Z0-9]*|0[8-9][0-9]* {printf("Lexical error - line %d : %s\n",yylineno,yytext);return LEX_ERR;}
```

3 语法分析

3.1 实验任务

分别在给出的语法分析器框架的基础上，实现一个 Sysy 语言的语法分析器：

(1) 基于 flex/bison 的语法分析器(C 语言实现)

(2) 基于 flex/bison 的语法分析器(C++实现)

(3) 基于 antlr4 的语法分析器(C++实现)

以上任务任选一个完成即可。

3.2 语法分析器的实现

本关卡选用基于 flex/bison 的语法分析器(C 语言实现)。在实验中，需要将产生式转换为 bison 的语法规则，并为每个产生式写明语义动作以构建抽象语法树。在实现时，需要注意含有 Exp 的产生式需要改写为含有 Exp 和不含有 Exp 的产生式，终结符';'用相应的 Token 代号(SEMICOLON)取代。

在书写语义动作时，根据 AST 数据结构的字段，使用 new_node 函数创建抽象语法树。具体而言，type 属性填写为左部非终结符 Stmt，int_val 填写为语句类别，float_val 填写为 0，symbol 填写为 NULL，d_type 填写为 NonType。而 left、mid 和 right 属性需要根据不同产生式填写，当有 1 个孩子节点时使用 right，有两个孩子节点时使用 left,right，当有三个孩子节点时使用 left,mid,right。

具体代码如下：

```
Stmt: LVal ASSIGN Exp SEMICOLON { $$ = new_node(Stmt, $1, NULL, $3, AssignStmt, 0, NULL, NonType); }
  | Exp SEMICOLON { $$ = new_node(Stmt, NULL, NULL, $1, ExpStmt, 0, NULL, NonType); }
  | SEMICOLON { $$ = new_node(Stmt, NULL, NULL, $1, BlankStmt, 0, NULL, NonType); }
  | Block { $$ = new_node(Stmt, NULL, NULL, $1, Block, 0, NULL, NonType); }
  | IF LP Cond RP Stmt ELSE Stmt { $$ = new_node(Stmt, $3, $5, $7, IfElseStmt, 0, NULL, NonType); }
```

```
    | IF LP Cond RP Stmt %prec THEN { $$ = new_node(Stmt, $3, NULL, $5, IfStmt,
0, NULL, NonType); }
    | WHILE LP Cond RP Stmt { $$ = new_node(Stmt, $3, NULL, $5, WhileStmt, 0,
NULL, NonType); }
    | BREAK SEMICOLON { $$ = new_node(Stmt, NULL, NULL, NULL,
BreakStmt, 0, NULL, NonType); }
    | CONTINUE SEMICOLON { $$ = new_node(Stmt, NULL, NULL, NULL,
ContinueStmt, 0, NULL, NonType); }
    | RETURN Exp SEMICOLON { $$ = new_node(Stmt, NULL, NULL, $2,
ReturnStmt, 0, NULL, NonType); }
    | RETURN SEMICOLON { $$ = new_node(Stmt, NULL, NULL, NULL,
BlankReturnStmt, 0, NULL, NonType); }
    ;
```

4 中间代码生成

4.1 实验任务

在给出的中间代码生成器框架基础上完成 LLVM IR 中间代码的生成，将 Sysy 语言程序翻译成 LLVM IR 中间代码。

4.2 中间代码生成器的实现

本关卡需要实现 visit()方法，用于生成 StmtAST 类抽象语法树的 IR。

当 sType=ASS 时，结点类型为赋值语句，它有两个子节点：lVal 和 exp。其语义是将 exp 的值 store 到代表左值的变量地址。

在代码中，首先需要告诉 LVal 节点当前是赋值语句的左值，而不是表达式。

```
requireLVal = true;
```

然后 visit(lVal)，并从 recentVal 取左值的 Value。

```
ast.lVal->accept(*this);
auto lval = recentVal;
```

接着 visit(exp)，并从 recentVal 取右值的 Value。

```
ast.exp->accept(*this);
auto rval = recentVal;
```

然后需要检查赋值语句左值和右值的类型，如果必要需要进行类型转换。

```
if(lval->type_->tid_==Type::FloatTyID && rval->type_->tid_
==Type::IntegerTyID){
        rval=builder->create_sitofp(rval,FLOAT_T);
}else if(lval->type_->tid_==Type::IntegerTyID && rval->type_->tid_
==Type::FloatTyID){
        rval=builder->create_sitofp(rval,INT32_T);
}
```

最后调用 create_store 生成 store 指令即可。

```
builder->create_store(lval,rval);
```

5 目标代码生成

5.1 实验任务

在给出的代码框架基础上，将 LLVM IR 中间代码翻译成指定平台的目标代码：

(1) 基于 LLVM 的目标代码生成(ARM)

(2) 基于 LLVM 的目标代码生成(RISCV64)

以上任务任选一个完成即可。

5.2 目标代码生成器的实现

本关卡选用基于 LLVM 的目标代码生成(RISCV64)。需要完善的代码分为三部分，分别是初始化目标、指定目标平台、初始化 addPassesToEmitFile()的参数。

```
// 补充代码 1 - 初始化目标
InitializeAllTargetInfos();
InitializeAllTargets();
InitializeAllTargetMCs();
InitializeAllAsmParsers();
InitializeAllAsmPrinters();
// 补充代码 2 - 指定目标平台
auto target_triple = "riscv64-unknown-elf";
module->setTargetTriple(target_triple);
// 补充代码 3 - 初始化 addPassesToEmitFile()的参数，请按以下顺序
// (1) 调用 getGenFilename()函数，获得要写入的目标代码文件名 filename
std::string filename = getGenFilename(ir_filename, gen_filetype);
// (2) 实例化 raw_fd_ostream 类的对象 dest。
//      Flags 置 sys::fs::OF_None
//      注意 EC 是一个 std::error_code 类型的对象，你需要事先声明 EC，
std::error_code EC;
auto Flags = sys::fs::OF_None;
raw_fd_ostream dest(filename, EC, Flags);
//      通常还应在调用函数后检查 EC，if (EC) 则表明有错误发生(无法创建目标文件)，此时应该输出提示信息后 return 1
if (EC)
{
   errs() << "Could not open file: ";
   return 1;
}
// (3) 实例化 legacy::PassManager 类的对象 pass
```

```
    legacy::PassManager pass;
    // (4)  为 file_type 赋初值。
    auto file_type = gen_filetype == CGFT_AssemblyFile ? CGFT_AssemblyFile :
CGFT_ObjectFile;
```

6 总结

6.1 实验感想

本次实验紧扣理论知识，从认识基本编译工具到词法分析，从语法分析到语义计算，从中间代码生成到目标代码生成，无不加深了我对课堂知识的理解与体会。同时，本次实验的文档尤为详尽，有助于快速掌握各种编译器、机器指令架构、词法分析工具等使用方法。

6.2 实验总结与展望

本次实验中，我掌握了常见的编译工具，例如 gcc、clang、make；了解了两种机器平台架构——arm 和 riscv；学会了如何使用词法分析工具 flex 和语法分析工具 bison；掌握了语义计算、抽象语法树、中间代码生成等代码实现。

本次实验中，主要通过补全代码的形式完成关卡，难度较低，可以尝试提高关卡难度，以对编译系统有更好的认识。同时，可以设计一些关卡实现对抽象语法树的简单可视化工具加深理解。