编译原理实验报告

人员分工

匡俊骅

1. 词法分析部分
2. 语法分析部分（表达式、语句块等）
3. 符号表的创建与作用域的管理
4. 中间代码生成（架构搭建、中间代码类的创建、语句块的中间代码生成）
5. 汇编部分（和跳转有关的汇编部分生成）
6. 代码的合并与测试

薛鹏

1. 语法分析部分（声明、赋值等）
2. 语法树结构的创建，语法树的生成
3. 中间代码生成（变量、指针、数组声明语句的中间代码生成，声明变量偏移量的计算）
4. 汇编部分（和赋值有关的中间代码生成，整数的输出）
5. 代码的合并与测试

肖雄峰

1. 语法分析部分（数组、指针等）
2. 语法树的生成与绘制
3. 中间代码生成（条件语句、循环语句的中间代码生成）
4. 汇编部分（和跳转有关的中间代码生成）
5. 代码的合并与测试

郭耘赫

1. 词法分析部分
2. 语法分析部分（for循环，while循环，if和else等）
3. 中间代码生成（加减乘除取余表达式，赋值语句、布尔表达式以及回填，指针的地址计算）
4. 类型检查（指针类型和数组类型的检查报警）

5. 汇编部分（和表达式有关的中间代码生成）
6. 代码的合并与测试

项目架构

--AsmCode--

--AsmCode.h&&AsmCode.cpp--

c++文件，根据指令生成单条汇编语言

--AsmCodeGenerate.h&&AsmCodeGenerate.cpp--

c++文件，翻译三元式，并调用对应的Asmcode生成函数，生成对应汇编语言

--print_int_i.asm--

汇编语言文件，调用printf实现print函数

--result.asm--

生成的汇编语言文件

--interCode--

--InterCode.h&&InterCode.cpp--

c++文件，根据语法树生成四元式

--Node--

--BaseNode.cpp&&BaseNode.h--

c++文件，数据结构，以二叉树的方式保存，功能上模拟多叉树

--BTNode.h&&BTNode.tpp--

c++文件，可以根据树的生成函数，打印树

--NodeType.h--

c++文件，保存了节点的类型

--main--

可执行文件，测试BTNode的打印效果

--Symbol--

--Symbol.h&&Symbol.cpp--

c++文件，程序经过词法分析和语法分析生成了语法树，将语法树进行语义分析，确定变量的作用域。

--MakeFile--

编译指令文件，在Linux下可以直接执行

--c_compiler.l--

.lex 文件,包含需要识别的关键词，和对应的执行动作

--lex.yy.c--

c语言文件，lex执行.lex文件得到词法分析器的c语言版本

--c_compiler.y--

yacc文件，用于语法分析，其第三部分的main函数是依次完成文件读入、词法分析、语法分析（抽象语法树的生成）

运行方式

1. 进入项目目录
2. 将想要测试的代码写入到 `test.txt` 文件中
3. 在linux虚拟机下，在终端输入 `make` 指令，即可生成汇编文件 `result.asm` 以及可执行文件 `result`，并执行该文件。
4. 注意，test.txt 文件中的变量名不能声明为 t 开头的变量，因为 t 开头的变量在汇编时会当做临时变量处理
5. 注意，如果想要修改 `c_compiler.l` 文件，则需要在 `make` 之前单独执行语句 `lex c_compiler.l`，并将生成的 `lex.yy.c` 文件中的以下代码删除，替换为 `#define yywrap() 1`

```
#ifndef YY_SKIP_YYWRAP
#ifdef __cplusplus
extern "C" int yywrap ( void );
#else
extern int yywrap ( void );
#endif
#endif
```

主要思路

1. 词法分析：

使用词法分析工具flex，确定好符号表，和对应符号的标识符

2. 语法分析：

使用语法分析工具bison

生成语法树部分：

- 采用长子-兄弟的结构来构造语法树；
- 每个节点保存该节点的类型和内容；
- 在规约时进行节点的构建和节点之间的连接；

节点构建：

根据设计好的文法来确定节点类型即可

3.语义分析

- 根据节点类型确定是否要新增作用域、通过搜索对应作用域，确定是否是新增符号
- 符号通过unordered_map来保存，存储符号的类名是Symbol
- 作用域使用二叉树来保存，彼此关系有父子和兄弟两种，存储作用域的类名是SymbolArea

4.中间代码生成

使用四元式作为中间代码生成的结果。

- 通过一个QuadItem类来对四元式进行定义
- 类中包括op，arg1，arg2，result四个关键字段以及四元式的类型
- 参数和目标（arg1，arg2，result）使用共用体存储，以同时兼容立即数和变量

通过InterCode类来实现中间代码的生成、存储、回填等操作

- 对生成的语法树进行遍历操作
- 根据节点类型来生成不同的中间代码

下面举例说明各种不同的语句生成的具体方式

4.1 while 语句：

判断节点内容若为While_Statement,使用 Exp_Stmt_Generate对while语句的第一个孩子进行处理，生成两条的四元式，一条为真时的跳转，一条为假时的跳转，将真的其压入quad_list；处理完条件后对body部分进行处理，按顺序将生成的四元式放入quad_list，执行完while的body后,回填条件为假需要跳出循环后面的三地址码。

4.2 if语句

使用Exp_Stmt_Generate对if语句的第一个孩子进行处理，生成两条的四元式，一条为真时的跳转，一条为假时的跳转，将其压入quad_list；处理完条件后对body部分进行处理，按顺序将生成的四元式放入quad_list，在输出时根据四元式的操作对应格式输出。

4.3 if-else语句

处理方式与if语句类似，只是在处理else的部分时，需要加入全新的符号表，同时使用backpatch回填函数将quad_list语句中对应true和false的四元式的跳转目标分别设为body块在quad_list中对应的的起始下标。

4.4 普通表达式

根据表达式生成的语法树，生成加减乘除和取余运算相对应的代码

4.5 逻辑运算、关系运算、布尔表达式

- 节点为关系操作时，需要生成两条分别代表是true和false的四元式，true的四元式包括optype、arg1、arg2、result，false的四元式包含result和jump
- 节点为逻辑运算时，需要维护truelist和falselist，并根据“与”和“或”的不同使用merge函数进行合并操作
- 对于“非”操作，只需要将truelist中的内容和falselist的内容进行调换

4.6 数组（一维）和指针（一维）

对于数组变量、在进行定义时需要对栈中地址进行确认，对于指针变量，需要在其指向某个变量后确定栈内地址。

5.代码优化

在生成中间代码时，标记符号表中的变量是否被使用； 在生成中间代码后遍历之，删除未使用变量的定义四元式。

6. 错误处理

6.1 变量未定义

在变量使用前从作用域中查找四元式用到的变量是否存在于表中，如果在本作用域和上层的所有作用域中都不存在，那么报变量未定义的错误；

6.2 变量重复定义

在变量定义前，如果在本符号表中找到同名的变量则报变量重定义的错误

7.类型检查

- 在定义指针变量时，若赋值号右边的值为整数，报类型错误警告
- 在定义整数时，若赋值号右边为地址，报类型错误警告

8.汇编部分（代码生成）

- 在代码生成部分，使用nasm作为目标的汇编代码
- 在遍历之前，给每个可能被跳转的部分加入跳转标签四元式（给四元式打lable）
- 对于所有四元式来进行遍历操作
- 在中间代码部分，已经知道各个变量的栈中地址，所以可以在栈中根据地址把这个变量放到对应的位置，这样就可以实现基本的赋值、取值功能
- 对于赋值操作，使用mov
- 对于简单的加减乘除表达式运算，将arg1压入eax，之后调动运算操作，之后将eax的值导入结果
- 对于取模运算，先进行除法运算，再从edx中取出余数结果
- 对于变量的输出，可以将待输出的整数压入eax，然后调用C语言的printf函数，将eax和预先定义好的int_format作为参数传入

实验总结

遇到的问题与解决方法：

1. 如何在yacc中调用c++的语法。

> - 在yacc生成的y.tab.h中将用到的c++ .h头文件引入
> - 将lex生成的lex.yy.c中的以下代码删除，替换为 `#define yywrap() 1`
>
> ```
> #ifndef YY_SKIP_YYWRAP
> #ifdef __cplusplus
> extern "C" int yywrap ( void );
> #else
> extern int yywrap ( void );
> #endif
> #endif
> ```

2. 在进行语法分析、中间代码生成以及汇编代码部分时，代码是多人分工完成，在执行时遇到段错误之类的问题，无法立即确定问题所在。

> 我们在整合代码时，采取线下一起调试的方法，遇到问题能够及时的让写这段代码的人来修改，而不是让某个人去修改别人的代码。在后期整合汇编代码时，我们也是采用了线上腾讯会议的方式，共同调试。在整合的过程中，我们遇到了很多次找不到问题的定位的情况，最终采用print调试的方法才定位到了错误的语句。

3. 不了解汇编语言

> 队伍中并没有人对汇编有过了解，在生成汇编代码后，无法正常的打印输出，在查询各种资料以及向了解汇编的同学询问后，通过调用c语言的printf函数完成了汇编的输出。

4. 在执行case1的汇编代码时，执行到while语句，出现了死循环，而正常执行这段代码不应该出现死循环

> 经过gdb调试，发现在我们的代码中，如果存在一个变量a的值是-1，让这个值去对2取余，结果竟然是1。经过查阅资料，发现我们在取余时调用的是汇编的div指令，而div是无符号除法，因此造成了错误。将div指令改为有符号除法idiv指令后，最终通过了case1的测试

仓库地址

https://github.com/shinianzhiqian/c-language-compiler