实现简单编译器

高钰洋崔立骁

November 2020

目录

1 作业解析

我们本学期的编译原理课程即将结束了，作为结课的成果，也为了，大作业需要大家实现简单的 SySy 语言编译器。之前大家已经完成了词法分析，语法分析部分，也就意味着符号表，语法树已经构造完成。按照编译器的代码处理流程，我们还有如下几步需要实现。

- **类型检查**: 检查语法是否符合语法规则，编译器提示的错误信息多在此处产生
- **中间代码生成**：将目标语言翻译为机器无关代码，通常为三地址码
- **代码优化**: 使用代码优化方法 (如循环展开) 优化中间代码，提高程序性能。比如 gcc 根据不同的优化等级（O1,O2...）使用不同的代码优化策略
- **汇编代码生成**: 生成机器有关代码，也就是汇编代码

鉴于我们只需要实现简化编译器，因此大作业的基础实现中，我们直接跳过中间代码生成和代码优化，只进行**类型检查**和**汇编代码生成**两步，使用语法树直接翻译出汇编代码，汇编代码就可以执行在我们的电脑上了。实现最终的大作业将是本次编译原理课程最具成就感的一步，实现编译器的程序员就可以称为一个“浪漫”[1]的程序员了。

和之前的作业一样，我们提供了一个简单编译器的框架，代码库地址[2]。但该框架的功能远不足以完成最后的作业要求，因此还需要大家编写大量代码。大作业的评分标准是区分等级的，具体的评分等级划分在第4节说明。

你实现的编译器默认生成 AT&T 格式的 x86 汇编，我们也鼓励大家尝试生成 ARM 汇编，二者在实现难度上基本没有差距。关于 ARM 汇编的格式以及如何在 x86 平台上运行，请参考 lab2 相关文档。

2 类型检查

类型检查是编译过程的重要一步，以确保操作对象与操作符相匹配。每一个表达式都有相关联的类型，如比较运算表达式的类型为布尔型，而计算表达式的类型一般为整型或浮点型等。类型检查的目的在于找出源代码中

[1] 众所周知，程序员的三大浪漫是编译原理，操作系统和图形学

[2] https://github.com/lixiao-c/lab6_example

不符合类型表达式规定的代码，在最终的代码生成之前报错，使得程序员根据错误信息对源代码进行修正。

实现类型检查程序，首先要规定一些基本类型。在本次作业的代码实现中，一类最简单的基本类型包括整形 (Interger)，布尔型 (Boolean) 以及无类型 (Notype) 等。无类型会在无法确定表达式类型时用来暂时填充。

类型检查可以在建立语法树的过程中完成。我们需要为语法树结点增加一项以表示该节点代表的表达式类型。类型检查也是自底向上的过程，父节点需要检查孩子结点的类型，并根据孩子结点类型确定自身类型。有一些表达式可以在语法制导翻译时就确定类型，比如整数就是整形，这些表达式通常是语法树的叶结点。而有些表达式则依赖其子表达式确定类型，这些表达式则是语法树中的内部节点。举例而言，对于加法运算，其类型是整形还是浮点性取决于其子表达式，而如果其子表达式类型为字符型，则发生了类型错误。

我们以 while 为例，给出了类型检查代码的框架。type_check 函数会在每次生成语法树结点时调用。具体到 while 语句，其第一个孩子，也就是小括号中的循环判断条件必须是 bool 型，否则则报错退出。这里给出的代码没有完成为结点赋予类型的部分（如上文提到的为加法表达式赋予整形），需要同学们自行补充。一些复杂语句的类型检查也会比较复杂，如作业基本要求中的 if 语句的类型检查，还有进阶要求中的函数类型检查等，这些需要大家自己好好思考一下。

```
void tree::type_check(Node *t)
{
	// type check, write your own code here
	if (t->kind == STMT_NODE)
	{
		if (t->kind_kind == WHILE_STMT)
			if (t->children[0]->type != Boolean)
			{
				cerr << "Bad boolean type at line: " << t->lineno << endl
				exit(1);
			}
			else
				return;
	}
	/* ... */
	return;
}
```

3 代码生成

代码生成要求将上次作业生成的语法树通过遍历生成目标代码，目标代码要求为汇编语言程序。当然目标代码的生成需要一些辅助函数的帮助，比如 label 的生成，临时变量的计数等等，当然同学们也可以根据自己的要求实现一些辅助函数来帮助代码的生成。如下例所示，我们在 main 函数中调用了 get_label() 和 gen_code 函数来分别实现生成 label 和生成汇编代码的任务。

```
int main(int argc, char *argv[])
{
  yyin = fopen(argv[1], "r");
  ostream* asm_out = new ofstream(argv[2]);
  yyparse();
  parse_tree.get_label();
  parse_tree.gen_code(*asm_out);
  return 0;
}
```

3.1 label 生成

我们知道在遇到分支条件语句的时候，程序会根据条件语句的真假执行不同的语句，在汇编语言中则会根据条件得值进行跳转，跳转到不同的标号所在的语句。但是生成目标代码的程序是如何知道应该跳转到的 label 的值多少呢？这就需要我们在生成汇编代码之前遍历语法树，提前给相应的结点生成编号。

在 get_label 函数中，可以通过判断节点的不同类型进行不同的处理。

这里给出一个 while 语句生成 label 的简单的例子，需要注意的是，这里的例子是十分不完整的，目的是为大家提供一些思路，需要大家做很多额外的补充。

例子中我们需要首先获得 while 语句的两个孩子，分别是条件语句与循环体，之后我们根据循环语句的逻辑多起 label 进行设置。其中 new_label() 函数的作用是生成一个新的 label。

```
void tree::stmt_get_label(Node *t)
{
   switch (t->kind_kind)
   {
   case WHILE_STMT:
```

```
        {
            Node *e = t->children[0]; //循环条件
            Node *s = t->children[1]; //循环体

            if (t->label.begin_label == "")
                t->label.begin_label = new_label();

            //循环体的下一条语句————循环的开始（继续循环）
            s->label.next_label = t->label.begin_label;

            //循环体的开始标号即为循环条件的真值的标号
            s->label.begin_label = e->label.true_label = new_label();

            //循环结束的标号
            if (t->label.next_label == "")
                t->label.next_label = new_label();

            //循环条件的假值标号即为循环的下一条语句标号
            e->label.false_label = t->label.next_label;

            //兄弟节点的开始标号即为当前节点的下一条语句的标号
            if (t->sibling)
                t->sibling->label.begin_label = t->label.next_label;
            //递归生成
            recursive_get_label(e);
            recursive_get_label(s);
        }
    /* ... */
    }
}
```

与 while 语句的类似，其他语句的 label 生成也可以采用类似的思路，需要同学们自己思考和实现。

3.2 汇编代码生成

在生成完 label 后我们可以进行汇编代码的生成（当然同学们可以根据自己的实现进行一些其他的处理如生成中间代码等），汇编代码的生成也是一个对于语法树的遍历的过程，根据结点类型进行相应的输出。这里也给出一个 while 语句的简单的框架。

```
void tree::stmt_gen_code(ostream &out, Node *t)
{
    if (t->kind_kind == WHILE_STMT)
    {
```

```
        if (t->label.begin_label != "")
            out << t->label.begin_label << ":" << endl;
        recursive_gen_code(out, t->children[0]);
        recursive_gen_code(out, t->children[1]);
        out << "\tjmp " << t->label.begin_label << endl;
    }
    else if (....)
    {
        /* ... */
    }
    /* ... */
}
```

当遇到 while 语句的时候，我们首先输出该语句的 lable 标号，之后递归遍历该节点的两个孩子（循环条件与循环体），最后跳转回循环体开始的 label。

4 评分标准

本次大作业的评分标准是分级别的，下面将进行详细的评分标准说明。注意，实验指导书提供的测试样例可能无法包含所有情况，所以仅仅通过这一个测试样例不代表就完成了相应的级别要求，我们在检查作业时会改动测试样例以检查编译器实现的完整性。为了大家尽可能完成的检查自己实现的编译器，我们提供了一个自动测试程序[3]，目前包含几十个测试样例，也可以自行添加。

4.1 类型检查（满分 2 分）

要求：完成级别一中规定内容的类型检查。你需要支持变量未声明、重声明错误检查，类型错误检查（比如字符串、布尔值不能参与某些运算、输入输出函数也有参数类型要求）等。大家需要准备一份包含类型错误的代码，以说明你完成的类型检查的情况。我们会随机改动你的输入源代码，以验证类型检查的正确性。进阶要求所需要的类型检查不计算在这部分的分数中。

4.2 级别一要求及测试样例（满分 11 分）

要求：

[3] https://github.com/gilsaia/lab6_test

1. 数据类型：int，char, 常量字符串。
2. 变量声明：不要求实现多作用域
3. 语句：赋值（=）、表达式语句、语句块、`if`、`while`、`for`、`return`.
4. 表达式：算术运算（+、-、*、/、%，其中 +、-都可以是单目运算符）、关系运算（==, >, <, >=, <=, !=）和逻辑运算（&&（与）、||（或）、!（非））
5. 注释.
6. 简单的输入输出（依 SysY 运行时库或自行定义词法、语法、语义均可，最好可以支持有“格式控制符”的 `printf`, `scanf`）。

```
/*
   I'm level 1 test.
 */
void main() {
   int a, s;
   a = 10;
   s = 0;
   char ch;
   scanf("%c", &ch);
   printf("%c", ch);
   while(a>0 && a<=10 || a%100==10) {
      a -= 1;
      a = 10;
      s += a;
      if(s > -10) {
         printf("result is: %d\n", s);
         int b;
         b = 10;
         int i;
         for(i=0; i<b; i++) {
            printf("Have fun: %d\n", i);
         }
      }
   }
}
// No more compilation error.
```

4.3 级别二要求及测试样例（满分 1.5 分）

进阶要求的每一项，我们会根据**实现难度**、**实现效果给予分数**。也就是说，每一个级别，在保证一定的工作量前提下，你可以只实现你最喜欢（或最有挑战性）的一部分。

要求：

1. 实现多作用域的变量声明
2. 支持每种类型的 `const` 常量的声明（与初始化），对于 `int/char` 类型，支持十进制、八进制、十六进制数。同时支持变量的定义与初始化。
3. 支持任意维数组。
4. 支持结构体类型。
5. 支持指针，支持引用。

对于语法分析，你将需要实现常量声明与初始化、数组、结构体声明与使用、对指针等语法结构的支持。对各进制数在词法分析中作识别。为了支持非基本数据类型，你需要修正你的类型系统。

类型检查作业中，你需要检查常量的未初始化错误，对常量的重赋值错误。对于非基本类型，实现关于它们的类型检查。

下面是一个参考的测试样例，具体检查时大家可以根据实际的实现情况进行改动或重写。有兴趣的同学可以参考 Sysy 语言的测试库。

```
/*
   I'm level 2 test. Without pointer.
 */
struct Matrix {
   int id;
   int arr[10][10];
} m1, m2, m3;
const int len = 10;
void main() {
   /*scope test*/
   int a, b;
   b = 10;
   scanf("%d", a);
   if (a > 5){
      int b = 5;
```

```
        b += a;
        printf("%d\n", b);
    }else{
        while(a < 5){
          int b = 1;
          a -= b;
          printf("%d\n", a);
        }
    }

    /* struct and array test*/
    m1.id = 1, m2.id = 2, m3.id = 3;
    for(int i=0; i<len; i++) {
        for(int j=0; j<len; j++) {
            m1.arr[i][j] = i;
            m2.arr[i][j] = j;
            m3.arr[i][j] = m1.arr[i][j] + m2.arr[i][j];
        }
    }

    for(int i=0; i<len; i++) {
        for(int j=0; j<len; j++) {
            printf("<%d>[%d][%d] %d\t",m3.id, i, j, m3.arr[i][j]);
        }
        printf("\n");
    }

}
```

4.4 级别三要求及测试样例（满分 1.5 分）

要求：

1. 支持 `breakcontinue`
2. 支持任意数量基本类型参数的函数。此项为必须项。
3. 支持对运行时库中函数的调用。

对于语法分析，你主要需要实现对**有基本类型参数的函数**、对 `break/continue` 语句的支持。

类型检查作业中，你需要实现对函数调用作参数检查以及对 `break/continue` 语句作 `within loop` 检查。此外，鉴于函数本质也是变量，也要被纳入类型检查中。

```
/* I'm level 3 test, With no runtime */
int s = 0;
int f(int x, int y) {
    s += x*x + y*y;
    return s;
}
void main(){
    int i=0;
    int a=1, b=1;
    int line;
    scanf("%d", &line);
    if(line > 10000) line = 10000;
    while(true) {
        if(f(a++, b++)<line) {
            printf("sum is: %d\n", s);
        } else {
            printf("result is:%d\n", s);
            break;
        }
    }
}
```

4.5 级别四要求（视实现情况酌情加分）

大作业的最终级别！

1. 你需要支持中间代码生成（可以想象，你需要额外设计 CFG 的数据结构，它由三地址码构成；并设计算法完成 AST 到 CFG 的转换。）
2. 并在其上实现代码优化。优化借助计时函数衡量。