编译原理

实验（五）中间代码生成器

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2023年6月5日
学　　院	信息学院
课程名称	编译原理

实验（五）中间代码生成器

目录

1 实验目的

掌握中间代码生成器的构造原理和编程方法。

2 实验内容

用自顶向下方法或Yacc进行语法分析的基础上，编写一个中间代码生成程序。

3 运行结果

运行结果如图 1，该程序会在控制台输出结果的同时，将结果写入`generate.txt`中。完整输出内容见附录：完整输出。

```
100:    i = 1
101:    _t1 = (float)0
102:    sum = _t1
103:    flag = false
104:    if i <= 100 goto 114
105:    goto 106
106:    _t2 = (float)3000
107:    if _t2 < sum goto 109
108:    goto 112
109:    _t3 = (float)5000
110:    if sum < _t3 goto 114
111:    goto 112
```

图 1: 运行结果

4 支持break

为了支持`break`，程序需要进行以下的操作：

- 检查`break`是否在循环中，如果不在循环中，则报错。
- 生成一条中间代码，跳转到正确的位置。

考虑到循环天然具有嵌套的特性，因此自然地可以使用栈来记录当前是否在循环中。为了能够正确入栈，考虑到YACC采取的是自底向上的分析方法，因此需要在`for`和`while`的产生式中加入一个空产生式$Q \to \epsilon$进行入栈操作。

空产生式$Q \to \epsilon$的动作中，程序会执行如下操作：

```
Q       :
        {   break_lists[tos++] = NULL;  }
```

`break_lists`是用于记录`break`位置同时反映嵌套循环的栈。在Q产生式被使用时，程序会将`break_lists`中入栈一新的元素，等待后续的产生式将其设为正确的`break`位置。进而，循环语句DO和$WHILE$的产生式也需要分别修改成$stmt \to DO\ Q\ M\ stmt\ldots$和$stmt \to WHILE\ Q\ M\ldots$。这样一来，使用这些产生式进行分析时就会导致$Q$被使用。

在循环结束后，程序会将栈顶出栈，然后加入循环的$nextlist$中。这样一来，就可以和其他产生式一起正确地生成中间代码了。

在入栈出栈操作之外，程序还需要在break的产生式中生成一个goto中间代码，它负责跳转到正确的位置。由于break的产生式中，break的位置已经被记录在break_lists栈中，然后栈顶又会加入到循环的*nextlist*中，因此这条goto中间代码的位置也会在进行回填时被一同正确地确定，无需进一步额外处理。

以while循环为例，处理它的动作如下：

```
| WHILE Q M '(' bool ')' M stmt
{
    backpatch(table, $8.n_list, $3.addr);
    backpatch(table, $5.t_list, $7.addr);
    $$.n_list = merge_goto_list($5.f_list, break_lists[--tos]);
    gen(table, slist, clist, jmp, -1, -1, $3.addr);
}
```

而break的动作如下：

```
| BREAK ';'
{
    $$.n_list = NULL;
    if (tos == 0)
        yyerror("\"break\" statement doesn't match any loop");
    else
    {
        break_lists[tos - 1] = merge_goto_list(break_lists[tos - 1], new_goto_list(table->size
        gen(table, slist, clist, jmp, -1, -1, -1);
    }
}
```

这些动作完全实现了上述逻辑。其中merge_goto_list是额外实现的一个函数，它的作用是将两个链表合并成一个。

5 实验总结

在本次实验中，我通过在基于YACC的语法分析器上进一步实现一个中间代码生成器，进一步加深了对编译原理的理解。同时，也加深了对中间代码生成一章中的知识的体会，更深刻地理解了书中所给出的翻译方案，并亲自动手实现了它们。

这是本学期最后一次实验，在本学期的实验中，我从词法分析器开始，逐步从手写的词法分析器过渡到基于LEX的词法分析器，然后再到基于YACC的语法分析器，最后，综合以上学习到的工具，以及语法制导定义、语法制导翻译方案的各项知识，综合应用完成了此次中间代码生成器的实验。虽然在实验过程中遇到了一些困难，但是在实验中，我学习到了如何使用工具生成编译器，更获得了将理论知识应用到实践中的经验。

虽然以前有过实现编译器的经验，但是经过一学期的课程，我将自学的、碎片化的知识系统化、完善化，同时也学会了很多新的知识，例如在实践过程中被我忽略的自底向上的语法分析，以及在实践中一直在使用，但是没有系统、全面认识的语法制导定义、语法制导翻译方案等等。在一学期的过程中，我应用了大学三年以来学习到的几乎所有知识来完成实验，又对照我以前的经验，进行进一步的总结、提升和修正。这一学期的实验和理论学习确实让我受益匪浅，收获颇丰。

最后，感谢老师和助教老师们一学期以来的答疑解惑和辛苦付出！

A 附录：完整输出

```
-------- constant list --------
val        type        addr        width
-------------------------------
'y'        char        3026        1
2        int        3022        4
true        bool        3021        1
5000        int        3017        4
3000        int        3013        4
100        int        3009        4
false        bool        3008        1
0        int        3004        4
1        int        3000        4
-------------------------------


-------------- quadtable --------------
addr        op        arg1        arg2        result
---------------------------------------
100        movi        3000        -1        1000
101        itof        3004        -1        1013
102        movf        1013        -1        1004
103        movb        3008        -1        1012
104        jle        1000        3009        114
105        jmp        -1        -1        106
106        itof        3013        -1        1021
107        jlt        1021        1004        109
108        jmp        -1        -1        112
109        itof        3017        -1        1029
110        jlt        1004        1029        114
111        jmp        -1        -1        112
112        jeq        1012        3021        135
113        jmp        -1        -1        114
114        itof        1000        -1        1037
115        addf        1004        1037        1045
116        movf        1045        -1        1004
117        addi        1000        3022        1053
118        movi        1053        -1        1000
119        itof        3013        -1        1057
120        jlt        1004        1057        125
121        jmp        -1        -1        122
122        itof        3017        -1        1065
123        jgt        1004        1065        125
124        jmp        -1        -1        134
125        movc        3026        -1        1073
126        movb        3021        -1        1012
127        invi        3000        -1        1074
```

```
128         eq         1073         3026         1078
129         not         1078         -1         1079
130         btoi         1079         -1         1080
131         addi         1074         1080         1084
132         movi         1084         -1         1000
133         jmp         -1         -1         135
134         jmp         -1         -1         104
---------------------------------------
```

```
100:         i = 1
101:         _t1 = (float)0
102:         sum = _t1
103:         flag = false
104:         if i <= 100 goto 114
105:         goto 106
106:         _t2 = (float)3000
107:         if _t2 < sum goto 109
108:         goto 112
109:         _t3 = (float)5000
110:         if sum < _t3 goto 114
111:         goto 112
112:         if flag == true goto 135
113:         goto 114
114:         _t4 = (float)i
115:         _t5 = sum + _t4
116:         sum = _t5
117:         _t6 = i + 2
118:         i = _t6
119:         _t7 = (float)3000
120:         if sum < _t7 goto 125
121:         goto 122
122:         _t8 = (float)5000
123:         if sum > _t8 goto 125
124:         goto 134
125:         c = 'y'
126:         flag = true
127:         _t9 = -1
128:         _t10 = c == 'y'
129:         _t11 = !_t10
130:         _t12 = (int)_t11
131:         _t13 = _t9 + _t12
132:         i = _t13
133:         goto 135
134:         goto 104
```