

语义分析

17341197 张富瑞

一.实验目的

- 1.实验目的：构造TINY + 的语义分析程序并生成中间代码
- 2.实验内容：构造符号表，用C语言扩展TINY的语义分析程序，构造TINY + 的语义分析器，构造TINY + 的中间代码生成器
- 3.实验要求：能检查一定的语义错误，将TINY + 程序转换成三地址中间代码

二.实验思路

经过前面两个实验，我们得到了由代码生成的语法树。现在我们可以通过对语法树后续遍历的方式，生成各个节点所对应的中间代码。当对于子节点的遍历完成后，也就是说子树的代码已经生成好了。从代码运行的角度来看就是“左部”和“右部”的结果已经在栈顶了。

1.PROGRAM

2.STMT_SEQUENCE

这两个节点则不需要生成代码，直接遍历子节点即可。

但是jump的指令需要我们对于指令进行链表化的处理，我们通过一个next指针来进行需要jump的指令的处理。

三.实验过程

```
MidCodeGenerator::MidCodeGenerator(TreeNode* root) {
    //start
    codes.push_back(link(MidType::START, "", "", ""));

    //中间代码生成
    generate(root);

    //end
    codes.push_back(link(MidType::END, "", "", ""));

    //处理jump
    for (MidCode* code : codes) {
        if (code->jump != -1) code->param3 = to_string(code->jump);
    }
}
```

使用后序遍历对根节点进行遍历，对于节点进行中间代码的生成。

```

switch (node->type) {
    //if-stmt -> if logical-or-exp then stmt-sequence [else stmt-sequence]
end

    case NodeType::IF_STMT: {
        node->children[0]->B = codes.size();
        generate(node->children[0]);

        //then
        int then_b = codes.size();
        generateMidCode(node->children[1]);
        MidCode* code = link(MidType::JUMP, "", "", "");
        codes.push_back(code);

        //else
        int else_b = codes.size();
        generate(node->children[2]);
        code->jump = codes.size();
        backPatch(node->children[0]->F, else_b);
        backPatch(node->children[0]->T, then_b);
        break;
    }
}

```

这里是if then else 语句的语法树节点的中间代码生成示例，这个节点会有三个子节点，分别递归调用函数，遍历节点。节点的B代表开始的行数，T代表为true的代码开始的地方，F为false的代码开始的地方。我们记录好then之后开始的代码在代码集中的位置以及else之后开始的代码在代码集中的位置，将子节点的T的jump指针指向then所代指的指针，将子节点的F的jump指针指向else所代指的指针。方便之后的jump操作。

```

int MidCodeGenerator::merge(int a, int b) {
    MidCode* p1 = codes[a];
    MidCode* p2 = codes[b];
    MidCode* p = p2;
    while (p2->next)p2 = p2->next;
    p2->next = p1;
    return p->pos;
}

```

在or表达式中，只要有一个节点的为T，or就为T，那么它True开始的代码的链表需要指向其中一个子节点的true链开始的地方，同时要将两个子节点的true链连接在一起。在and表达式中，与or节点相反。父节点的false链开始需要指向其中一个子节点的false链开始的地方。这里的merge就是用来连接子节点的代码。

```

void MidCodeGenerator::opJump(TreeNode* node, MidType type, const string
&param1, const string &param2) {
    node->T = codes.size();
    node->B = node->T;
    node->F = node->T + 1;
    codes.push_back(link(type, param1, param2, ""));
    codes.push_back(link(MidType::JUMP, "", "", ""));
}

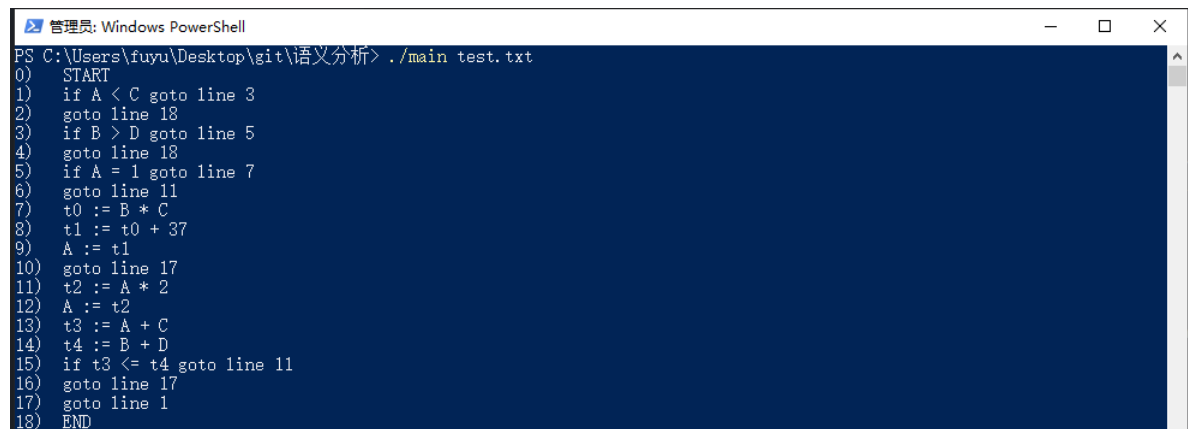
```

进行大小的比较的时候，我们生成的中间代码会有两个，一个是为true的时候的跳转，一个是正常的跳转，所以会添加两条中间代码。

四.实验结果

程序

```
int A,B,C,D;
while A<C and B>D do
    if A=1 then
        A:= B*C+37
    else
        repeat
            A:=A*2
        until (A+C)<=(B+D)
    end
end
end
```



```
PS C:\Users\fuyu\Desktop\git\语义分析> ./main test.txt
0) START
1) if A < C goto line 3
2) goto line 18
3) if B > D goto line 5
4) goto line 18
5) if A = 1 goto line 7
6) goto line 11
7) t0 := B * C
8) t1 := t0 + 37
9) A := t1
10) goto line 17
11) t2 := A * 2
12) A := t2
13) t3 := A + C
14) t4 := B + D
15) if t3 <= t4 goto line 11
16) goto line 17
17) goto line 1
18) END
```

程序

```
int x,fact;
read x;
if x>0 and x<100 then {don't compute if x<=0}
    fact:=1
    while x>0 do
        fact:=fact*x
        x:=x-1
    end
    write fact;
end
```

```
PS C:\Users\fuyu\Desktop\git\语义分析> ./main test1.txt
0) START
1) read x
2) if x > 0 goto line 4
3) goto line 16
4) if x < 100 goto line 6
5) goto line 16
6) fact := 1
7) if x > 0 goto line 9
8) goto line 14
9) t0 := fact * x
10) fact := t0
11) t1 := x - 1
12) x := t1
13) goto line 7
14) write fact
15) goto line 16
16) END
```