编译原理三实验报告

211220127顾嘉宇

实验功能

在本次实验中，我的任务为**要求3.2**，所以在编写代码时无需考虑结构体变量的定义和使用，当检测到结构体定义时直接报错。另一方面则要考虑数组作为函数参数、高维数组类型变量的访问、数组赋值等问题。因此， 在操作数中额外存储type(用来数组某一维度对应的elem)和op_addr(用来存储数组的某一维度第一个元素对应的地址)。

在词法分析，语法分析，语义分析均正确后，将输入的文件代码输出成中间代码。

操作数

```
struct Operand_{
    enum{
        OP_VARIABLE,    // 变量（var）
        OP_CONSTANT,    // 常量（#1）
        OP_LABEL,       // 标签(LABEL label1:)
        OP_TEMP,        // 临时变量（t1）
        OP_FUNCTION,    //函数
        OP_ARRAY,       //数组
        OP_GET_POINT,   //获得以目标操作数的值为地址的空间变量
        OP_GET_ADDR,    //获得目标操作数的值的地址
    }kind;
    union{
        int var_no; // 变量定义的序号
        int const_val; //操作数的值
        int label_no; // 标签序号
        int temp_no; // 临时变量序号（唯一性）
        char* func_name; //函数名字
        Operand op;     //目标操作数，对应GET_POINT和GET_ADDR
    } u;
    Operand op_addr;//地址操作数，专门给数组元素使用
    Type type; // 计算数组、结构体占用size
};
```

中间代码

```
struct InterCode_{
    enum{
        IC_LABEL,    // LABEL x :
        IC_FUNCTION, // FUNCTION f :
        IC_ASSIGN,   // x := y
        IC_PLUS,     // x := y+z
        IC_SUB,      // x := y-z
        IC_MUL,      // x := y*z
        IC_DIV,      // x := y/z
        IC_GOTO,     // GOTO x
        IC_IF_GOTO,  // IF x [relop] y GOTO z
        IC_RETURN,   // RETURN x
```

```
        IC_DEC,      // DEC x [size]
        IC_ARG,      // ARG x
        IC_CALL,     // x := CALL f
        IC_PARAM,    // PARAM x
        IC_READ,     // READ x
        IC_WRITE,    // WRITE x
    } kind;
    union{
    Operand op; //单操作数中间代码
    struct{Operand right, left; } assign; //等于
    struct{Operand result, op1, op2; } binop; //三地址代码
    struct{Operand result;int size;} dec; //数组声明
    struct { Operand result, op1, op2; char* relop; } ifgoto;  //IF_GOTO语句
    }u;
};
```

中间代码链表和变量链表

```
struct CodeList_{
    InterCode code; // 中间代码列表实现方式
    CodeList prev, next;
};

struct _Variable{ // 变量的实现方式
    char* name;
    Operand op;
    Variable next;
};
Variable var_head,var_tail; //分别指向变量链表的头尾
CodeList intercodes;//双循环中间代码链表，其中头结点为空
```

指令翻译

对输入文件进行分析，每生成一句中间代码，就将其追加到中间代码链表的尾部。在翻译完所有代码后，从中间代码链表的第二个节点开始，依次对每个节点中包含的信息进行对应的翻译，并将其输出到文件中，从而执行完整个程序。

选做任务

```
if(strcmp(root->children[1]->name,"LB") == 0){ //| Exp LB Exp RB
    Operand tmp1 = new_temp();
    Operand tmp2 = new_temp();
    Operand tmp3 = new_temp();
    translate_Exp(root->children[0],tmp1); //获取第一个Exp信息，即数组名称或高维数组
    translate_Exp(root->children[2],tmp2); //获取第二个Exp信息，即index
    int size = get_size(tmp1->type); //计算对应维度的单位偏移量大小
    Operand base = new_constant(size); //生成偏移量操作数
    add_interCode(new_InterCode(IC_MUL,tmp3,tmp2,base,NULL)); //生成中间代码，即
tmp3为索引乘上对应维度的单位偏移量
    add_interCode(new_InterCode(IC_PLUS,tmp3,tmp1->op_addr,tmp3,NULL));
    //将高于当前一个维度的数组的首个元素地址加上总偏移量
    place->kind = OP_GET_POINT; //Exp传入的place类型为指针型，即当前地址存储的值
    place->type = tmp1->type->u.array.elem; //类型为高维数组类型的后一个类型
    place->op_addr = tmp3; //地址即为前面计算出的地址
}
```

高维数组仍然采用递归的方式实现，先逐步分析到高维数组的最高维度，即对于$a[i][j][k]$，先分析到a，然后分析$[i]$中的索引，将数组a的第一个元素的地址加上索引乘上对应维度的单位偏移量，并将新生成的地址付给传入的place->op_addr，并将上一个维度对应type的elem赋给place，从而实现递归操作，依次添加对应的偏移量，完成对应元素的访问。

```
while (params!=NULL)
{// 构造函数参数列表判断参数类型（选做内容中需要支持结构体和数组作为参数）
    Operand operand;
    ...
    if(params->type->kind == ARRAY){
        operand = new_arr(params->name);
        add_interCode(new_InterCode(IC_PARAM,operand-
>op_addr,NULL,NULL,NULL));//声明数组时先声明变量地址
    }
    params = params->tail;
}
params = type->u.function->param;
while(params != NULL){//随后再次重新遍历，输出语句，声明变量地址和变量名的对应关系
    if(params->type->kind == ARRAY){
        Operand op = get_operand(params->name);
        add_interCode(new_InterCode(IC_ASSIGN,op,new_point(op-
>op_addr),NULL,NULL));
    }
    params = params->tail;
}
void translate_Args(struct Node* root, int write_func){
    ...
    Operand t1 = new_temp();
    translate_Exp(root->children[0],t1);
    ...//省略一系列if-else判断
    add_interCode(new_InterCode(IC_ARG,t1->op_addr,NULL,NULL,NULL));//传入数组地址
}
```

数组参数的实现只需要在检测到数组时先声明地址，然后将数组地址与数组对应即可，并在传参时传入数组地址。

编译&测试方式

进入`Code`文件夹所在路径，执行`make`命令，即可获得`parser`可执行文件。 执行`./parser test.cmm out.ir`命令，即可对一个待测试的源文件进行词法、语法、语义的分析。

实验感悟

本次实验是实现编译器的中间代码的翻译，使翻译出的中间代码可以在模拟器中正确地执行。这次实验中，生成中间代码本身的难度并不大，与第二次实验的遍历语法树节点类似。只需要利用好之前实验中的语法树和符号表所给的信息，再结合指导手册上给的提示即可生成正确的指令。