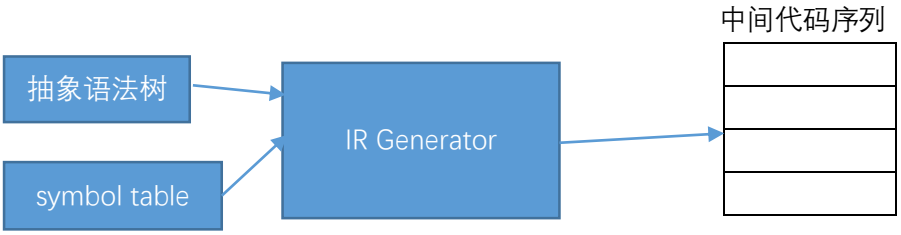
实验三　中间代码生成

姓名： 郑来栋
学号： 151220170

一. 中间代码生成

本次实验以前两次实验生成的语法树和符号表为输入，采用语法制导的翻译方案，将语法树翻译成中间代码



二． 实验进度

已完成全部要求（选做和必做要求），通过了课本上的测试用例（以及自己设计的一些测试用例）

三．数据结构

本实验的一个难点就在于如何设计合适的数据结构来表示不同类型的中间代码和不同类型的操作数

1．操作数有常量，变量等，但变量可以是普通变量，或者加了*或&的内存访问相关的变量，因此我的 Operand 的结构如下：

```
enum OPKIND {VARIABLE, CONSTANT, FUNC};
enum TYPE {ADDRESS, STAR, NORMAL};
struct Operand
{
    enum OPKIND kind;
    enum TYPE type;
    union{
        char *text;
        int constant;
    };
};
```

2. 不同类型的中间代码有不同的操作数类型，对应实验教材上的中间代码格式规范

```
struct IR{
    enum IRKIND kind;
    union{
        struct {struct Operand *op; } oneop;
        struct {struct Operand *var; int size; } dec;
        struct {struct Operand *dest, *src;} binop;
        struct {struct Operand *op1, *op2, *dest; }triop;
        struct {struct Operand *op1, *op2, *dest; char *relop;} condjump;
    };
};
```

3. 中间代码采用带附加头节点的双向链表表示

```
struct IRList
{
    struct IR *ir;
    struct IRList *prev;
    struct IRList *next;
};
```

四. 实现过程中的一些难点

1. 函数定义形参和函数调用实参的顺序问题

函数定义的形参(PARAM)和函数调用的实参列表的顺序必须相反， 如 f(int a[5], int b)，则形参表将生成

FUNCTION f :

PARAM a

PARAM b

而对 f 的调用则按下面的顺序（x 传递给 b， y 传递给 a）

ARG x

ARG &y

...CALL f

这个只需要注意 translate_Args 的时候，采用链表的头插法即可

2. 函数参数可以传递一维数组，传递方式为引用传递

举个例子，比如下面的代码序列：

```
int f(int a[10][11])
{
    f(a)
}
int main()
{
    int b[10][11];
    f(b)
}
```

这里产生一个问题， 在 main 函数中调用函数 f，将 b 作为参数传递(传递的是 b 的地址)：

ARG &b

CALL f

在函数 f 的开头，声明参数 v1（v1 代表参数 a，与我的实现方式有关）

PARAM v1

接下来 f 递归调用自己，首先对 Exp->ID 进行翻译，查符号表，发现 a 是一个数组，于是乎生成下面的中间代码：

ARG &v1

CALL f

这就产生了问题，v1 本就应该是一个地址，不能再加&操作。解决这个问题的方法就是在 translate 函数定义的时候，将当前函数的形参列表都记下来，之后遇到 ID，如果是在参数列表中的，就不再生成取地址前缀

3．如何开始？

面对这样一个不算小的作业，先写哪里？这是一个比较困难的问题。是先设计数据结构？还是先进行分析打印到 stdout，分析正确再设计数据结构，这些都会影响到实验进度。当然，翻译过程是按照 C—文法，根据语法制导的工具进行翻译。不管怎么说，还是有点难度的，自己也学会了很多新的 C 语言工具，如 static 关键字，C 语言如何进行合理的抽象等。