L4实验报告

1 实现功能

在前三次实验的基础上，将输入的C--源代码翻译为中间代码。

2 实现方法

根据实验需要，在语义分析阶段的函数表中先加入两个函数，分别是write(num)和read(),用来表示三地址代码的输出和输入。

```
void addFunc(){
    Function read = new Function_();
    read->returnType = new Type_();
    read->returnType->kind = Type_::BASIC;
    read->returnType->u.basic = INT;
    funcMap.insert({"read", read});


    Function write = new Function_();
    write->returnType = new Type_();
    write->returnType->kind = Type_::BASIC;
    write->returnType->u.basic = INT;
    field out;
    out.name = "output";
    out.type = new Type_();
    out.type->kind = Type_::BASIC;
    out.type->u.basic =INT;
    write->varLi.push_back(out);
    funcMap.insert({"write", write});
}
```

然后就是经典的从语法分析生成的语法生成树的根节点开始，对整个语法树进行深度优先遍历，并在需要时生成相关代码。具体的实现在ir.cpp文件中。

由于生成的中间代码需要写入到名为out1.ir的文件中，所以我在翻译时先把所有中间代码加到一个string out1中，当翻译完成后，就得到了一个大的string，最后将这个string输出到文件中。

```
void getFile(char* path){
        ofstream fout;
        fout.open(path,ios::out);
        fout<<out1<<endl;
        cout<<out1;
        fout.close();
}
```

定义了一个新的map<string, int>symbols来存储翻译过程中已经用到的遍量。

定义了getSize(Type t)来计算数组的elem和结构体中的域的大小。

具体的实现大致是参考实验教材上的思路，在它的基础上做了一些补充和完善。

3 设计

translate_Exp函数有时需要一个string place参数，有时则不需要。

不太想重载这个函数。于是定义一个不可能用到的值：nullPlace来占位，在不需要传入place参数时使用这个无意义的值，这样即可不用考虑参数的问题。(仅适用于较短的代码)

```
string nullPlace  = "t1000000";
```

4 遇到的BUG

本次实验没有遇到太多问题，但还是有一个小bug找了很久。

刚开始输出的中间代码总有一部分是重复的，还以为是递归或者函数调用的问题，未果。于是考虑是字符串拼接的问题。

在翻译时是把新生成的代码直接拼接到out1后面，形如：

```
out1=out1+"PARAM "+newVar(temp.at(i).name)+"\n";
```

不过我有一处写成了out1+=out1.....，所以会出现重复代码。