思路

学号：19373440 姓名：王雨飞

开始前的分析

1.在lab8之前，我在调用函数(part 6)以及lab7中都提前为lab8做了准备，虽然知道lab8难，但是已经有了之前准备好的数据结构，写起来应该不会那么难了。

2.在之前建立的`BVarMap`对于函数中的局部变量处理同样适用。这个`BVarMap`指的是当前Block的局部变量(即被{ }包括的代码部分)，有一点不一样的地方是：对于我符号栈的实现方式(也就是Block中嵌套更多Block)，在不同的函数中，不仅需要清空`BVarMap`，也需要清空每一层的`VarMap`。

3.文法的修改也有很多，用我之前的思路，先去尽量正确的实现语法分析，保证在正确的情况下能跑代码。

4.这么多语义约束并不能看一遍就全记住，决定在写代码的时候，时不时来看一眼。

5.关于传参为数组的情况：因为在lab7就已经为了挑战实验中的多维数组准备，所以这里也需要额外的考虑，保证什么维度的数组都能用。

6.这个样例给的让人感觉完全不够，好多我想到的需要输出的`LLVM IR`语法，在样例中我都看不到。我只能尝试着自己去理解怎么正确使用`LLVM IR`语法 **(写实验报告的时候已经写完代码了，在这一条上我至少花了5个小时的时间来各种修改输出，连输出的逻辑都改过过好几次，很让人头秃)**。

写代码

1.先花了一整个下午来完成语法分析部分。这个lab就像lab7一样，虽然看上去文法修改没有很多，但是很多修改都关联到之前的判断，导致需要不断的debug来找自己语法分析的错误。

2.如果结构体里有`string`，不能用`malloc`分配内存，这个是踩坑了(对C++不够熟悉)。花了很多时间，也试过不在结构体里面使用`string`类型，但这会导致很多其他的代码都要改变写法(C++中`char*`和`string`是需要相互转化的，我个人感觉是不方便的)。后来还是在网上查资料，发现用`new`就可以解决这个问题。

完成代码

1.主要用到的额外代码如下：

```
struct FuncItem            //在lab3已经定义了，在lab8中添加了更多的属性
{
    int RetType;         //函数返回类型 1为int 0为void
    string funcName;     //LLVM IR中的函数名
    int paramsNum;       //函数参数个数
    string tfuncName;    //编译器中用的函数名(用来做判断等操作)
    vector<int> params;  //每个参数的类型 其中1为int 2为int*
    vector<int> paramsDimension;  //每个参数对应的维数
};
struct FuncItem *tempFuncItem;  //用来定一个函数
bool varIsParam;  //在LVal中使用，表示这个变量是函数的参数
int paramSt = 0; //在LVal中使用，用来定义当前的FuncRParam参数是哪个
bool funcHasRet = false; //在funcDef中使用，表示当前定义的函数有没有显式返回值
bool hasMain = false; //表示是否已经定义主函数
```

```
bool isAssignExp;      //表示当前的Exp是=后面的
map<string, struct FuncItem> FuncMap;
map<string, struct FuncItem>::iterator funcIt; //Funcmap变量迭代器
```

2.就像之前的每个lab一样，编译器读到哪个位置，比如说在读函数参数时，读到就输出对应的LLVM IR语句，其实没什么好说的。在我的实验报告最上面那个部分中的第6条，提到了我改了好几次输出LLVM IR语句的逻辑，但是我也记不得我每次的修改是什么(因为样例都过不去，不会`push`到`github`)

3.基本的定义函数的方式如下，这是函数`FuncDef()`中：

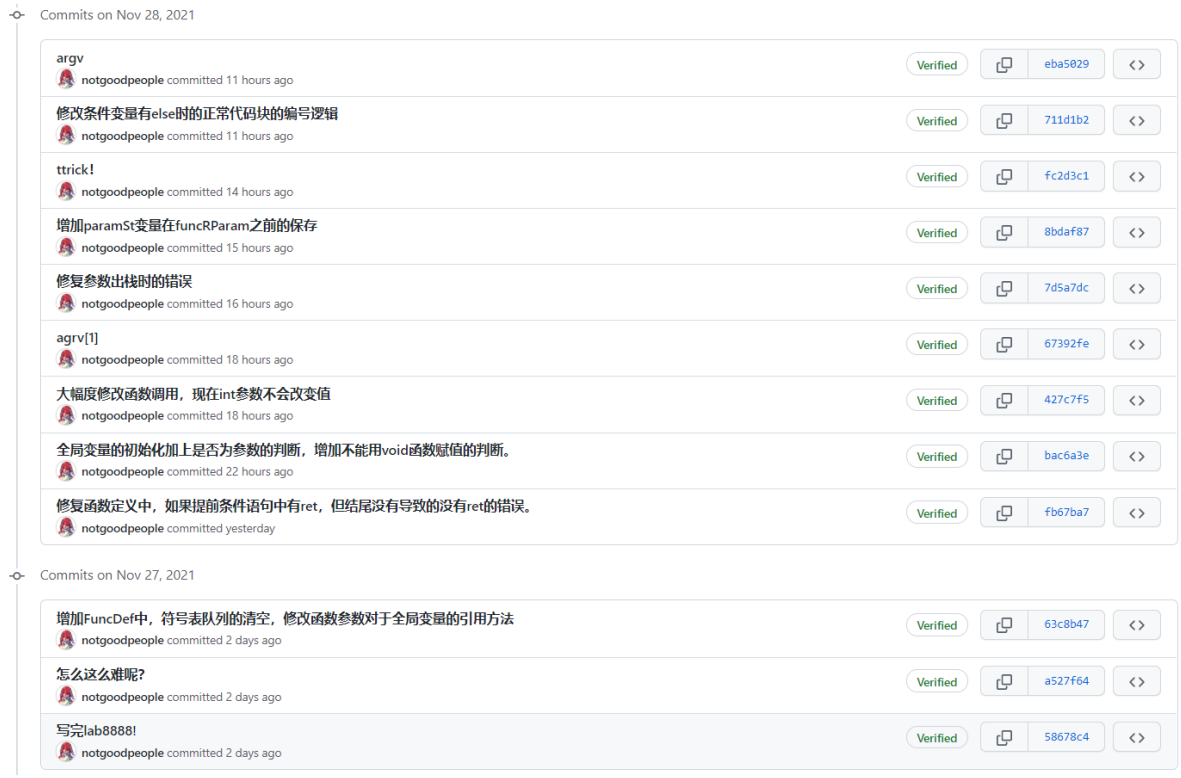
```
FuncDef():
tempFuncItem = new FuncItem;  //就是上面那个结构体
VarMapList.clear();      //清空了所有符号栈(一个Func中可以有多个Block嵌套，我都是存在List中)
BVarMap.clear();         //情况当前的局部变量Map
/* 一系列操作为tempFuncItem中的属性赋值，包括调用FuncFParams()等操作*/
```

4.篇幅有限，其它方面就不细说了。

Debug

1.我曾经以为lab7是我debug最难的一次，结果lab8就打破了lab7的记录。仅时间来说，大概花了一天半。感觉lab8的测试点很难，直接让我查出了很多历史遗留问题，而且改起来也很费劲。

2.github上debug的记录：(数量多且时间久)



3.首先是，在`FuncDef()`仅清空了`BVarMap`，而没有清空每一个`Block`一层一层连起来的`变量map`的队列，就相当于符号栈只清空了当前的Block，所以会导致很多读变量的错误。

4.函数参数对于全局变量的引用，主要还是实际写代码的时候考虑不周，有些情况考虑不到。其实这个改起来也挺麻烦的。

5.有些函数里面，只有`if else`语句，也就是说，不在`if`里`return`，就会在`else`中`return`。但是(也许我输出代码块的方式的问题)，LLVM IR并不智能识别是不是一定能`return`，所以我就根据函数返回类型，判断函数结尾有没有ret，没有就输出一下(int的情况下，返回什么都无所谓，我干脆就`ret i32 0`了)。

6.在我的代码中，作为参数的变量和普通变量的处理方法并不是完全分开的，一个很不好的地方就是我在写代码的时候忘记在某个该分类讨论的地方没有分类讨论。

7.不能用返回值为void的函数给变量赋值。

8.因为对`LLVM IR`的语法理解不够理想(我自己本来的输出和样例输出有些不同)，导致语法上出现了问题(比如说作为参数的i32或i32*应不应该修改值)。我印象中lab8的样例里，没有在函数中，对`i32`的参数进行修改的例子，所以我就出问题了。对应图中那个“大幅度修改函数调用”

9.参数出栈部分的代码部分写错了。

10.对我来说最难的地方，我发现lab4的代码逻辑写的有些问题。最后其实做的修改比较巧，没有太大的代码量。但是为了想这个比较巧的修改方式，也大概花了1个多小时硬想，看着错误的代码不停尝试。我手搓的当时过不去的代码如下：

```
int main(){
    int a=5;
        if(a>4){
            putint(4);
        }
        else{
        if(a==6){putint(6); }
        if(a==7){putint(7); }
        if(a==8){putint(8);}
        if(a==9){putint(9);  }
        a=a+1;
        }
    putint(a);
    return 0;
}
```

当时的错误是：第一个if的true块，认为下一个正常代码块是if(a==7)，所以不可避免的一定会执行a=a+1。这个细讲起来可能又得是1000多字，所以不细说了。

11.改完以后20/20。直接把野的代码扔去跑了一下挑战实验中的多维数组，错了2个。

感想

不算挑战实验的话，就是完结撒花了！感谢助教和老师为编译lab做的努力，我觉得这个实验做的真的非常成功，保证难度的同时也让人很好上手。(我也听说6系的舍友说他们的编译实验并不是很友好)。

我经历了好几次重构代码，也在好几个难度高的lab上饱受折磨，但是也确实让我能很好的体会编译的原理，也大幅增强了我写代码的能力(也熟悉了C++的数据结构)。