Lab8实验报告

201180024 陈灵灵

功能&实现

函数参数

- 允许数组作为函数的参数
- 数组仅包含一维数组

考虑一维数组作为函数参数的情况，本次实验对之前写的一些方法进行了修改。

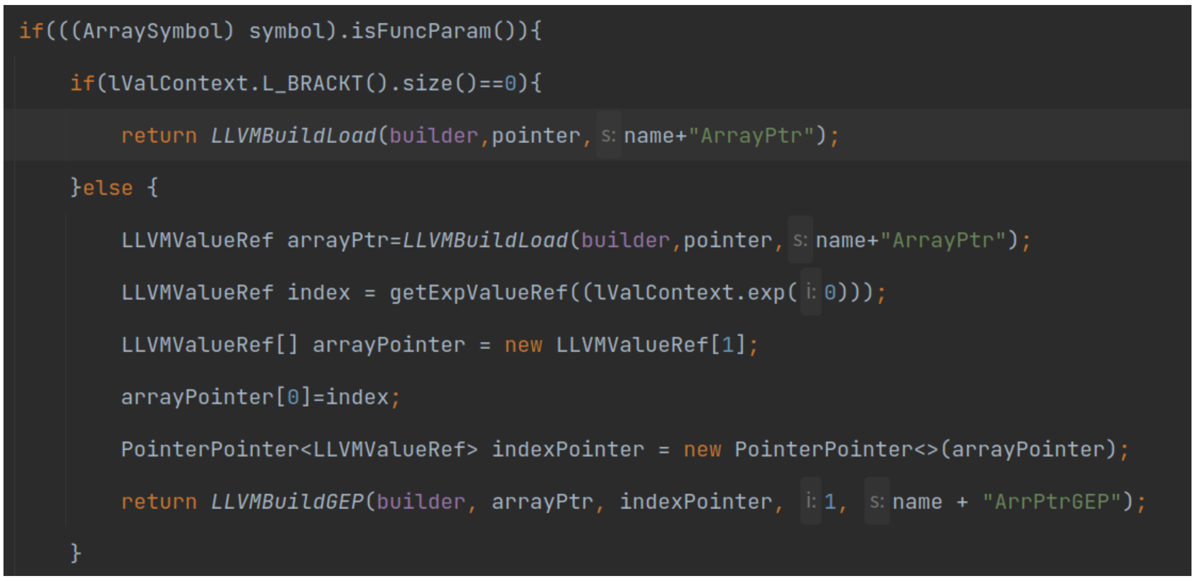
```
@Override
    public LLVMValueRef visitFuncDef(SysYParser.FuncDefContext ctx) {
        //函数参数类型
        PointerPointer<Pointer> argumentTypes = new PointerPointer<>(paramNum);
        for (int i=0;i<paramNum;i++){
            //TODO
            if(ctx.funcFParams().funcFParam(i).L_BRACKT().size()==0) {
                argumentTypes.put(i, i32Type);
            } else {     // array
                LLVMTypeRef pointerType = LLVMPointerType(i32Type, 0);   //类型为
i32*
                argumentTypes.put(i,pointerType);
            }
        }
        //...
        for(int i=0;i<paramNum;i++){     //形参
            if(ctx.funcFParams().funcFParam(i).L_BRACKT().size()==0) {
                //...
            }
            else{   //array
                LLVMTypeRef pointerType=LLVMPointerType(i32Type,0);
                LLVMValueRef pointer=LLVMBuildAlloca(builder, pointerType,
paramName+"FuncPtr");    //i32**
                LLVMValueRef argValueRef = LLVMGetParam(function, i);    //i32*
                LLVMBuildStore(builder, argValueRef, pointer);
                ArraySymbol arraySymbol=new ArraySymbol(paramName,pointer,true);
//pointer是i32**类型
                functionSymbol.define(arraySymbol);
            }
        }
    }

private LLVMValueRef getLValPointer(SysYParser.LValContext lValContext){
    else if(symbol instanceof ArraySymbol){     //array
            LLVMValueRef pointer=symbol.getPointer();
            if(((ArraySymbol) symbol).isFuncParam()){   //是函数参数,pointer为i32**
类型
                if(lValContext.L_BRACKT().size()==0){   //如果是a,则应该返回i32*类
型，所以需要Load
```

```
                    return LLVMBuildLoad(builder,pointer,name+"ArrayPtr");  //
                }else { //a[index]，Load后得到i32*类型，再GEP,注意i32*类型GEP的参数与
arrayType不同
                    LLVMValueRef
arrayPtr=LLVMBuildLoad(builder,pointer,name+"ArrayPtr");
                    LLVMValueRef index = getExpValueRef((lValContext.exp(0)));
                    LLVMValueRef[] arrayPointer = new LLVMValueRef[1];
                    arrayPointer[0]=index;
                    PointerPointer<LLVMValueRef> indexPointer = new
PointerPointer<>(arrayPointer);
                    return LLVMBuildGEP(builder, arrayPtr, indexPointer, 1, name
+ "ArrPtrGEP");
                }
            }else {     //不是函数参数，pointer为i32*类型
                if(lValContext.L_BRACKT().size()==0){   //如果是a,则应该返回指针类
型，用GEP0获得对应指针
                    LLVMValueRef[] arrayPointer = new LLVMValueRef[2];
                    LLVMValueRef zero = LLVMConstInt(i32Type, 0, 0);
                    arrayPointer[0] = zero;
                    arrayPointer[1] = zero;
                    PointerPointer<LLVMValueRef> indexPointer = new
PointerPointer<>(arrayPointer);
                    return LLVMBuildGEP(builder, pointer, indexPointer, 2, name
+ "GEPPtr");
                }else {     // a[index]
                    //...
                }
            }
}

private LLVMValueRef getExpValueRef(SysYParser.ExpContext expContext){
    else if(expContext instanceof SysYParser.LValExpContext){
            LLVMValueRef pointer=getLValPointer(((SysYParser.LValExpContext)
expContext).lVal());
            if(symbol instanceof ArraySymbol && ((SysYParser.LValExpContext)
expContext).lVal().L_BRACKT().size()==0){  //如果是a,则不需要做任何处理，返回i32*类型，
即数组的指针
                return pointer;
            }else {
                return LLVMBuildLoad(builder, pointer,/*varName:String*/name);
//a[index],则Load得到对应的值
            }
        }
}
```

Bug



```
if(((ArraySymbol) symbol).isFuncParam()){
    if(lValContext.L_BRACKT().size()==0){
        return LLVMBuildLoad(builder,pointer, s: name+"ArrayPtr");
    }else {
        LLVMValueRef arrayPtr=LLVMBuildLoad(builder,pointer, s: name+"ArrayPtr");
        LLVMValueRef index = getExpValueRef((lValContext.exp( i: 0)));
        LLVMValueRef[] arrayPointer = new LLVMValueRef[1];
        arrayPointer[0]=index;
        PointerPointer<LLVMValueRef> indexPointer = new PointerPointer<>(arrayPointer);
        return LLVMBuildGEP(builder, arrayPtr, indexPointer,  i: 1,  s: name + "ArrPtrGEP");
    }
```

`getLValPointer()`方法中，如果symbol是函数参数,则得到的pointer为i32**类型，一开始我误以为是i32 *类型，没有对它进行Load，导致后续操作出现类型不匹配。