1. 程序实现功能

将main函数和表达式翻译为LLVM IR。

2. 如何实现

2.1 重写访问函数定义的方法

按照使用手册，生成函数并创建基本块block。后续生成的指令都追加在block之后。

```
@Override
public LLVMValueRef visitFuncDef(SysYParser.FuncDefContext ctx) {
    LLVMTypeRef returnType = i32Type;
    PointerPointer<Pointer> argumentTypes = new PointerPointer<>(0);
    LLVMTypeRef ft = LLVMFunctionType(returnType, argumentTypes,/* argumentCount
*/ 0, /* isVariadic */ 0);
    LLVMValueRef function = LLVMAddFunction(module,
/*functionName:String*/"main", ft);
    LLVMBasicBlockRef block = LLVMAppendBasicBlock(function,
/*blockName:String*/"mainEntry");
    LLVMPositionBuilderAtEnd(builder, block);
    return super.visitFuncDef(ctx);
}
```

2.2 重写访问return语句的方法

return语句的语法为RETURN (exp)? SEMICOLON。如果exp的类型为number，那么直接使用LLVMConstInt构造函数返回值；如果是其他类型，那么函数返回值是访问exp节点的返回值。

```
@Override
public LLVMValueRef visitStmt8(SysYParser.Stmt8Context ctx) {
    if(ctx.exp() instanceof SysYParser.NumContext) {
        LLVMValueRef res = LLVMConstInt(i32Type,
convert(((SysYParser.NumContext) ctx.exp()).number().INTEGR_CONST().getText()),
/* signExtend */ 0);
        LLVMBuildRet(builder, res);
    }else{
        LLVMValueRef llvmValueRef = visit(ctx.exp());
        LLVMBuildRet(builder, llvmValueRef);
    }
    return super.visitStmt8(ctx);
}
```

2.3 重写访问运算符的方法

以访问三个单目运算符（'-', '!', '+'）为例。首先访问单目运算符的文法为unaryOp exp，其中unaryOp : PLUS | MINUS | NOT。先判断exp类型，是number则创建一个LLVMConstInt作为valueRef，不是则访问exp节点，其返回值是valueRef。如果unaryOp是'+'，那么返回valueRef自身；如果是'-'，那么借助LLVMBuildSub，用0减去valueRef得到其相反数；如果是'!'，则利用LLVMBuildICmp判断valueRef是否为0，是0的话则利用异或和扩展操作返回1，不是则返回0。

```
@Override
    public LLVMValueRef visitUnaryOpExp(SysYParser.UnaryOpExpContext ctx) {
        SysYParser.ExpContext exp = ctx.exp();
        LLVMValueRef valueRef;
        if(exp instanceof SysYParser.NumContext){
            valueRef = LLVMConstInt(i32Type,
convert(((SysYParser.NumContext)exp).number().INTEGR_CONST().getText()), 0);
        }else{
            valueRef = visit(ctx.exp());
        }
        if(ctx.unaryOp().PLUS()!=null){
            return valueRef;
        }else if(ctx.unaryOp().MINUS()!=null){
            return LLVMBuildSub(builder, zero, valueRef, "inverse");
        }else if(ctx.unaryOp().NOT()!=null){
            LLVMValueRef tmp_ = LLVMBuildICmp(builder, LLVMIntNE, valueRef,
zero, "tmp_");
            tmp_ = LLVMBuildXor(builder, tmp_, LLVMConstInt(LLVMInt1Type(), 1,
0), "tmp_");
            return LLVMBuildZExt(builder, tmp_, i32Type, "tmp_");
        }
        return super.visitUnaryOpExp(ctx);
   }
```

3. 遇到的困难及bug

3.1 单目运算符'!'

因为对LLVM的API还不是很熟悉，在判断值是否为0时卡了一会儿。LLVM有个API叫LLVMBuildNot，但我试了一下，它实现的功能和实验要求的取反符号是不同的。后来发现这个符号在助教给的使用手册里有示例😁。

3.2 八进制及十六机制数

一开始没有考虑八进制和十六机制数。