Stage-1实验报告

吴垒 2020010916

###### 实验内容

大致阅读并了解了完整的程序结构，理解了代码部分函数及变量、属性用途。

将minidecaf-tests仓库接入作业仓库。

由于Python框架下词法分析、语法分析内容已由框架完成，step2、step3、step4需完成的主体部分为从抽象语法树生成三地址码的内容，该部分代码主要存在于frontend/tacgen/tacgen.py，通过visit\*函数访问原始程序中\*节点并填充至抽象语法树。需要补全的就是visitUnary、visitBinary中将符号翻译为抽象语法树上伪汇编指令的过程，仅需在其中的词典内添加相应的键值对就行了。

需要注意的是step4中部分操作无对应汇编指令，因此在由三地址码生成汇编代码时无法处理对应指令，需要对其进行分类讨论，在backend/riscv/riscvasmemitter.py中RiscvAsmEmitter类下RiscvInstrSelector类的visitBinary方法下手动将一条伪指令拆为2-3条指令。同时还需要注意区分逻辑与、或与算数与、或，riscv支持的指令仅有算数与、或，我在助教更新测例后才意识到自己没有区分，又进行了修正，在utils/tac/tacop.py下BinaryOp中添加了LAD、LOR伪指令作为其子属性，并在utils/tac/tacinstr.py的Binary下添加了对应指令符号翻译。

###### 思考题

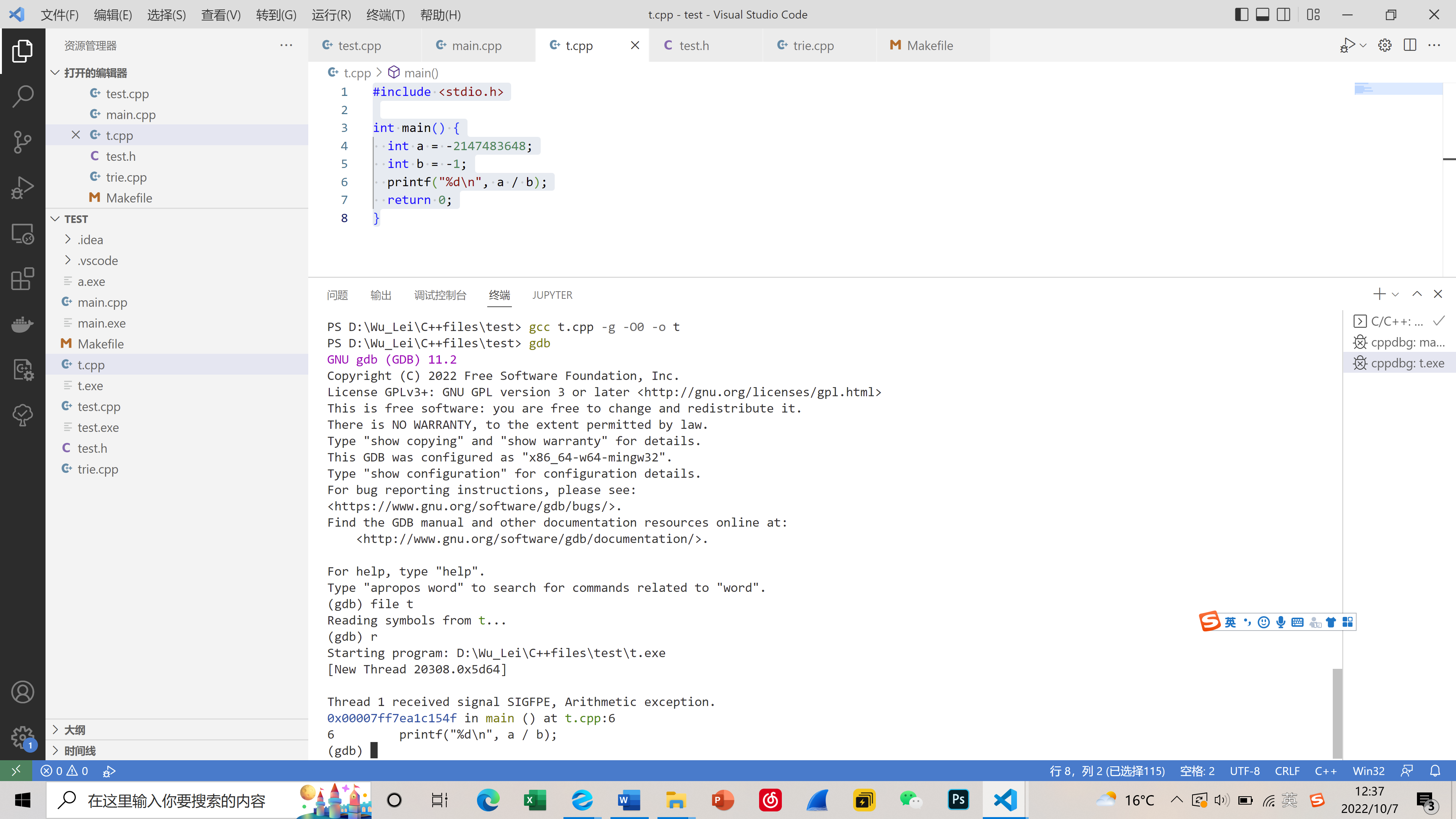
Step2：运算越界

-~2147483647 = -(-2147483648) = 2147483648（越界，原本应得到）=-2147483648（实际）

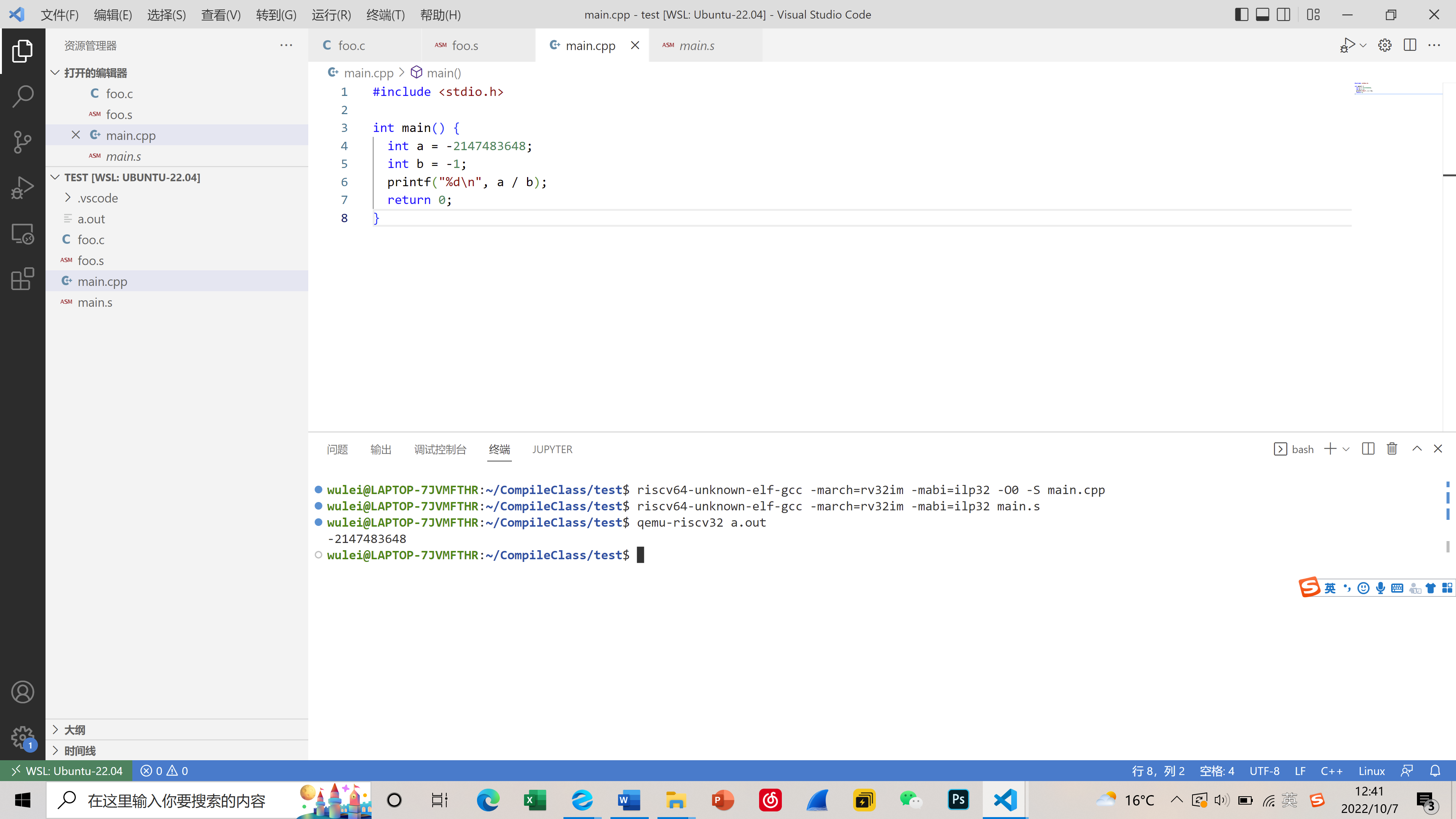
Step3：除法未定义行为

由上一个step可以猜测是负数相除造成的数组越界，即a = -2147483648, b = -1，原本应得到2147483648，但猜测会发生越界产生-2147483648。

运行结果：（本地X86-64架构，O0编译，gdb调试）（signal SIGFPE, Arithmetic exception.）



运行结果：（RISCV-32 qemu模拟器）（程序正常运行，输出错误结果-2147483648）



Step4：短路求值的好处？

短路求值指作为“&&”和“||”操作符的[操作数](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E6%95%B0?fromModule=lemma_inlink)[表达式](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F?fromModule=lemma_inlink)在求值时，只要最终的结果已经可以确定是真或假，求值过程便告终止。我认为该做法有以下两点好处：

加快运算速度。当已知前一表达式结果可确定整个表达式值时，可省去计算后一表达式的时间，在后一表达式较为复杂、运算时间较长（前后的运算时间不对称）时尤为明显，最终可加快整个代码的运行速度。

降低代码冗余度。如下例：while (++i < len(str) && a[i] != 0)，如果不支持短路求值则这句话需要放在循环中使用两层if、并以break的形式跳出，否则当i = len(str)时后一项发生数组越界，会导致程序报错；但在短路求值情况下该情况不会出现，因为前一式值为假时不会对后一式求值。

避免复杂的三目运算。如下例：a = Judge ? 5 : 10可改写为：

tmp = (Judge && (a = 5)) || (a = 10)

###### 借鉴内容

实验思路指导与问答墙