作业四：数据流分析框架（二）

作业准备

为了便于进行程序分析的相关实验，设计了一种简单的编译器中间语言 Toy IR。Toy IR 为 LLVM IR 的简化版。和 LLVM IR 类似，Toy IR 为控制流图 IR，并采用虚拟寄存器内存模型。为了让同学们关注于程序分析的核心思想与方法，Toy IR 的语法、指令集、类型系统均较为简单。

程序结构

Toy IR 的最顶层结构为模块，一个模块包含若干函数的定义。一个函数的函数体由至少一个基本块组成，函数入口指向函数体的第一个基本块，包含返回指令的基本块指向函数出口。一个基本块内为一系列顺序排列的指令，其中最后一条指令必须为控制流指令（跳转或返回）。

指令集

目前 Toy IR 仅包含下列六种形式的指令：

1. 赋值：例如%y = %x（将%x 的值赋给%y）或%x = 2（将 2 赋值给%x）。
2. 二元运算：如%y = add %x 2（将%x 与 2 相加并将结果写入%y）。目前支持 add（加），sub（减），mul（乘），div（除），rem（取余）五种二元运算。
3. 比较运算：如%y = gt %x 2（若%x 大于 2，将 1 写入%y；否则，将 0 写入%y）。目前支持 eq（等于），ne（不等于），lt（小于），le（小于等于），gt（大于），ge（大于等于）六种比较条件。
4. 直接跳转：如 jmp B1（跳转到标签为 B1 的基本块）。
5. 分支跳转：如 br %x B1 B2（如果%x 为 1，跳转到标签为 B1 的基本块；如果%x 为 0，跳转到标签为 B2 的基本块）。
6. 返回：如 ret %x（将%x 的值返回）。

值与类型

Toy IR 中，指令的输入输出数据称为值，一个值可以为一个局部变量，也可以为一个常量。Toy IR 的值仅有两种可能的类型：i32（32 位带符号整型）和 i1（布尔类型）。各种指令的类型约束为：

1. 赋值指令两侧的值必须为 i32 类型；
2. 二元运算指令的两个输入值、一个输出值必须为 i32 类型；
3. 比较运算指令的两个输入值必须为 i32 类型，输出值必须为 i1 类型；
4. 分支跳转指令的第一个输入值必须为 i1 类型。

由于各种指令的类型约束均非常明显，在代码中均不会标注其类型。

作业内容

本次作业的编程部分可以选择 C++/Java/Python 中任意一种语言进行编程。

注意：对于题目里要求打印输出的，请在作业文档里给出具体的输出结果。可以对命令行进行截图，也可以将输出结果复制到文档中，但均不得篡改原始输出结果。

一、**编写程序**对龙书图 9.10 所对应的 Toy IR 代码（见附录）进行**活跃变量分析**，具体要求为：

1、首先对该段代码进行**语法分析**，**构建控制流图**。Toy IR 语法较为简单，使用基本的 IO、字符串函数即可分析，不需要编写复杂的分析算法。附录里给出了 Toy IR 的 EBNF 语法规则作为参考，语法分析程序不需要严格遵照此语法，能支持本次作业即可。

2、然后**编写 Worklist 算法**，对控制流图进行分析，打印输出每个基本块的 def、use、IN、OUT 集合。

二、完成龙书习题 9.4.1，具体要求为：

1、**编写程序**，检查 Toy IR 代码中是否存在使用未定义（初始化）变量的情况。为减少不必要的工作量，请在第一题代码的基础上进行扩展来完成此题，尽可能复用已有代码。

2、编写存在变量未定义就使用情况的**示例 Toy IR 代码**，输入所编写的检查程序，**打印输出**存在未定义就被使用情况的**变量名**及使用未定义变量的**指令位置**。

对上述两个问题，请在作业文档中给出**代码实现的思路与方法**，以及程序的**输出结果**。

评分依据

思路、方法、结果正确；报告条理清晰、内容完整；代码组织合理、可读性强。

加分项

在实现分析算法时使用**位向量/位集合：**所有的集合编码为位向量**，**算法运行时通过位运算来进行集合运算**，**在算法结束时进行解码。位向量可以使用现有的库，如 C++的 boost::dynamic_bitset（不要使用 std::bitset）、Java 的 BitSet、Python 的 bitarray。

提交要求

作业文档使用 PDF 格式，所有代码放在名为 src 的目录内，将 PDF 文档和 src 目录打包为 ZIP/RAR 提交。项目工程文件、第三方库、编译结果等均不要提交。

常见问题

1. 附录里所示的 Toy IR 代码分支跳转指令的条件是常量，可以将其简化为直接跳转吗？

   答：不可以。将常量条件的分支跳转简化为直接跳转是常量传播所完成的工作，常量传播不在此次作业的范围之内。若简化，就会改变原有程序结构，所得结果和第三次作业手工计算的结果会不同。

2. 需要考虑输入的 Toy IR 代码有语法、语义（除了变量未定义就使用）上的错误吗？

   答：不需要。可以假定输入程序在语法、语义上是正确的。

3. 需要考虑函数调用的情况吗？

   答：不需要。目前课程并未介绍过程间分析，本次作业仅考虑单个函数，即 main。

4. 第二题第二小问的 “指令位置” 怎么理解：

   答：任何能够帮助定位问题指令的信息均可。可以输出代码行号，可以打印问题指令，也可以说明是哪个基本块的第几条指令。

附录

龙书图 9.10 的 Toy IR 代码

```
define i32 @main() {
B1:
    %a = 1
    %b = 2
    jmp B2
B2:
    %c = add %a %b
    %d = sub %c %a
    jmp B3
B3:
    %d = add %b %d
    br 1 B5 B4
B4:
    %d = add %a %b
    %e = add %e 1
    jmp B3
B5:
    %b = add %a %b
    %e = sub %c %a
    br 0 B6 B2
B6:
    %a = mul %b %d
    %b = sub %a %d
    ret 0
}
```

Toy IR 的 EBNF 语法规则（仅供参考）

```
# 词法规则
LABEL := [A-Za-z_]\w*
GLOBAL := @\w+
LOCAL := %\w+
CONST := [+-]?\d+
BIN := (add)|(sub)|(mul)|(div)|(rem)
COND := (eq)|(ne)|(lt)|(le)|(gt)|(ge)
TYPE := (i1)|(i32)

# 语法规则
ModuleDef := FuncDef+
FuncDef := `define` TYPE GLOBAL ParamList FuncBody
ParamList := `(` `)`
FuncBody := `{` BlockDef+ `}`
BlockDef := LABEL `:` InstDef+
InstDef := AssignInst
    | BinInst
    | CmpInst
    | JmpInst
    | BrInst
    | RetInst
AssignInst := LOCAL `=` Value
BinInst := LOCAL `=` BIN Value Value
CmpInst := LOCAL `=` COND Value Value
JmpInst := `jmp` LABEL
BrInst := `br` Value LABEL LABEL
RetInst := `ret` Value
Value := LOCAL | CONST
```