CS143 Written Assignment 4 Solution

James Dong

Due: June 4th, 2024 EOD

1. (a) `Value`:

Address	Line Number (Method)
0x8020	⟨line 3⟩ (`value`)
0x8028	⟨line 4⟩ (`set`)

`MulOp`:

Address	Line Number (Method)
0x8030	⟨line 3⟩ (`value`)
0x8038	⟨line 4⟩ (`set`)
0x8040	⟨line 8⟩ (`operate`)

`AddOp`:

Address	Line Number (Method)
0x8048	⟨line 3⟩ (`value`)
0x8050	⟨line 4⟩ (`set`)
0x8058	⟨line 11⟩ (`operate`)

(b) Heap layout:

Address	Value	Meaning
⟨object `Main`⟩ + 0x0000	5	(class tag)
⟨object `Main`⟩ + 0x0008	4	(object size)
⟨object `Main`⟩ + 0x0010	0x8800	(dispatch ptr)
⟨object `Main`⟩ + 0x0018	⟨object `Stack`⟩	(`stack`)
⟨object `Stack`⟩ + 0x0000	4	(class tag)
⟨object `Stack`⟩ + 0x0008	5	(object size)
⟨object `Stack`⟩ + 0x0010	0x8000	(dispatch ptr)
⟨object `Stack`⟩ + 0x0018	⟨object `Value`⟩	(`head`)
⟨object `Stack`⟩ + 0x0020	`void`	(`tail`)
⟨object `Value`⟩ + 0x0000	1	(class tag)
⟨object `Value`⟩ + 0x0008	4	(object size)
⟨object `Value`⟩ + 0x0010	0x8020	(dispatch ptr)
⟨object `Value`⟩ + 0x0018	2	(`value`)

`IO` may be included as well, without penalty. (Note that we don't really know the layout of `IO`, as it is not specified, so it was not intended to be included.)

(c) Stack layout:

Address	Value	Meaning
0x7777fff8	0x7ffffff8	(saved frame pointer)
0x7777fff0	⟨object `Main`⟩	(argument 0 of `main`)
0x7777ffe8	0x2000	(return address of `main`)
0x7777ffe0	⟨object `IO`⟩	(local variable `io`)
0x7777ffd8	5	(local variable `num`)
0x7777ffd0	0x7777ffe8	(saved frame pointer)
0x7777ffc8	⟨object `Main`⟩	(argument 0 of `reduce`)
0x7777ffc0	⟨line 54⟩	(return address of `reduce`)
0x7777ffb8	⟨object `AddOp`⟩	(local variable `x`)
0x7777ffb0	⟨object `AddOp`⟩	(local variable `op`)
0x7777ffa8	⟨object `Value`⟩	(local variable `temp`)
0x7777ffa0	10	(local variable `lhs`)
0x7777ff98	1	(local variable `rhs`)
0x7777ff90	0x7777ffc0	(saved frame pointer)
0x7777ff88	1	(argument 2 of `operate`)
0x7777ff80	10	(argument 1 of `operate`)
0x7777ff78	⟨object `AddOp`⟩	(argument 0 of `operate`)
0x7777ff70	⟨line 35⟩	(return address of `operate`)

Full credit should be awarded if the frame pointer addresses are off by 8 (matching the behavior of PA4's stack.)

2. (a)

$$\frac{so, S_1, E \vdash e_1 \mapsto v_1, S_2 \qquad v_1 \neq \bot}{so, S_1, E \vdash \textbf{try } e_1 \textbf{ catch } e_2 \textbf{ yrt} \mapsto v_1, S_2} \text{ [Try-OK]}$$

$$\frac{so, S_1, E \vdash e_1 \mapsto \bot, S_2 \qquad so, S_2, E \vdash e_2 \mapsto v_2, S_3}{so, S_1, E \vdash \textbf{try } e_1 \textbf{ catch } e_2 \textbf{ yrt} \mapsto v_2, S_3} \text{ [Try-Catch]}$$

$$\frac{}{so, S_1, E \vdash \textbf{throw} \mapsto \bot, S_1} \text{ [Throw]}$$

$$\frac{so, S_1, E \vdash e_1 \mapsto \text{Int}(i_1), S_2 \qquad so, S_2, E \vdash e_2 \mapsto \text{Int}(i_2), S_3}{so, S_1, E \vdash e_1 + e_2 \mapsto \text{Int}(i_1 + i_2), S_3} \text{ [Plus-OK]}$$

$$\frac{so, S_1, E \vdash e_1 \mapsto \bot, S_2}{so, S_1, E \vdash e_1 + e_2 \mapsto \bot, S_2} \text{ [Plus-Fail1]}$$

$$\frac{so, S_1, E \vdash e_1 \mapsto v_1, S_2 \qquad so, S_2, E \vdash e_2 \mapsto \bot, S_3 \qquad v_1 \neq \bot}{so, S_1, E \vdash e_1 + e_2 \mapsto \bot, S_3} \text{ [Plus-Fail2]}$$

Note: the rules [Plus-Fail1] and [Plus-Fail2] may be have a more general condition than $so, S_1, E \vdash [e_1 \ / \ e_2] \mapsto \bot, S_2$, such as $so, S_1, E \vdash [e_1 \ / \ e_2] \mapsto v, S_2$ and $v \neq \text{Int}(i_1)$. This handles non-integer arguments to + as errors that can be caught. Both solutions should be accepted for full credit.

(b)

$$\dfrac{\dfrac{\dfrac{}{\vdash 2 \mapsto \mathrm{Int}(2), S}\,[\text{Int}] \quad \dfrac{}{\vdash \mathbf{throw} \mapsto \bot, S}\,[\text{Throw}]}{\vdash 2 + \mathbf{throw} \mapsto \bot, S}\,[\text{Plus-Fail2}] \quad \dfrac{}{\vdash 3 \mapsto \mathrm{Int}(3), S}\,[\text{Int}]}{\vdash \mathbf{try}\ 2 + \mathbf{throw}\ \mathbf{catch}\ 3\ \mathbf{yrt} \mapsto \mathrm{Int}(3), S}\,[\text{Try-Catch}]$$

(c) Simplicio's scheme can cause stack misalignment when jumping to the catch handler. For example, the following code prints the wrong value:

```
class Main inherits IO {
    main() : Int {
        out_int(1 + try 2 + throw catch 3 yrt)
    };
};
```

This generates the following pseudo-assembly:

```
    ; prolog...
    push 1
    push 2
    jmp catch
    pop r1
    pop r2
    add r1, r2
    push r1
    jmp tryend
catch:
    push 3
tryend:
    pop r1
    pop r2
    add r1, r2
    push r1
    pop r1
    ; call out_int(r1)
    ; epilog...
```

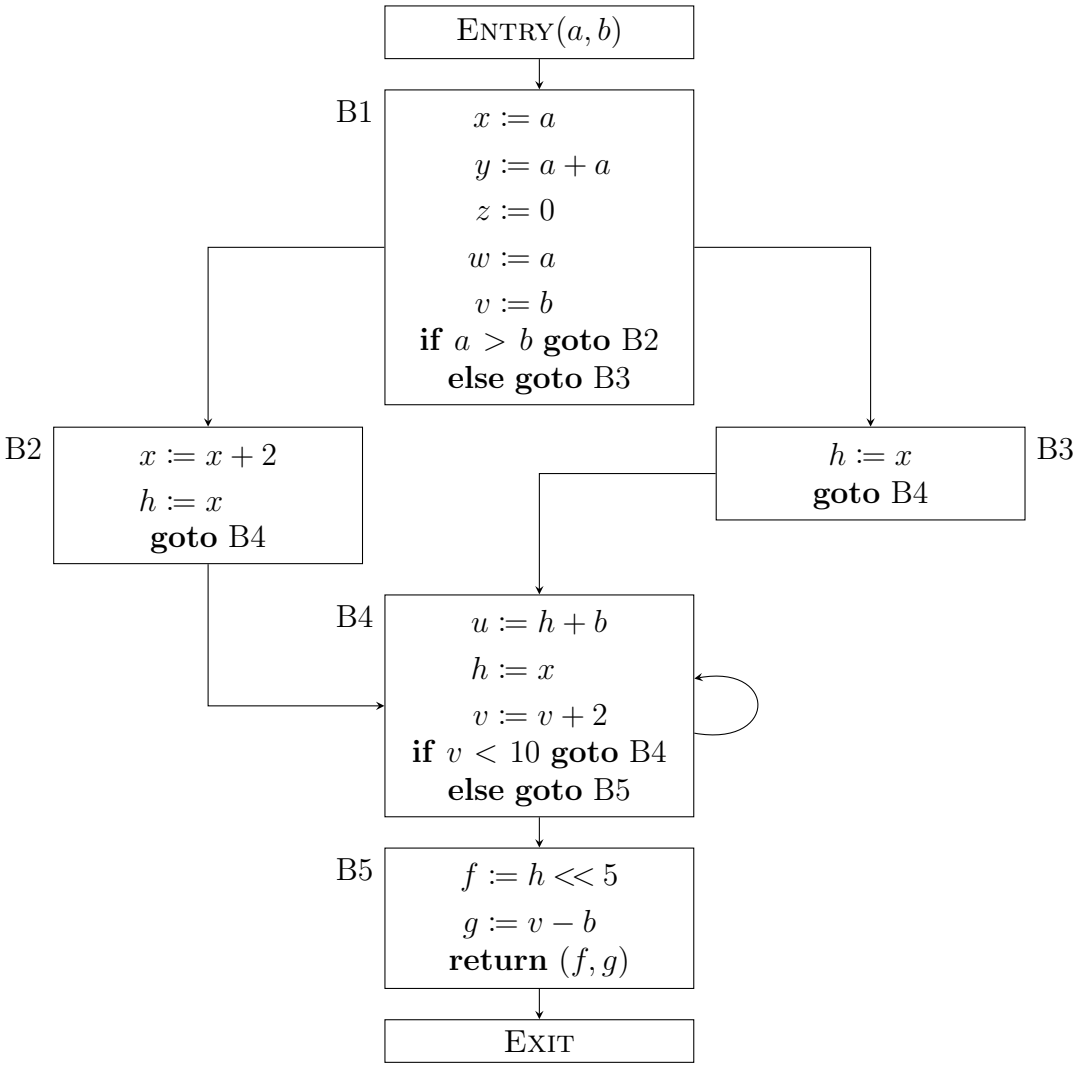
The catch handler only expects a stack depth of 1, since we are evaluating $1+\ldots$, but when evaluating the **throw** expression, we have pushed another value (2) onto the stack, since we were evaluating $2+\ldots$. Then the code following the try expression will pop the wrong value (2, instead of 1) and will wrongly produce $2+3=5$ instead of the correct result $1+3=4$.

Also, code that does *not* use a frame pointer can cause more serious issues, as this causes the entire activation record to be misaligned. For example, in the above code, there is an extra value (1) on the stack, which displaces the return value and likely causes the program to crash. This is avoided when there is a frame pointer, as resetting the stack pointer to the frame pointer resolves this misalignment. (Solutions that identify this problem without stating the additional assumption that a frame pointer is not used will receive partial credit.)

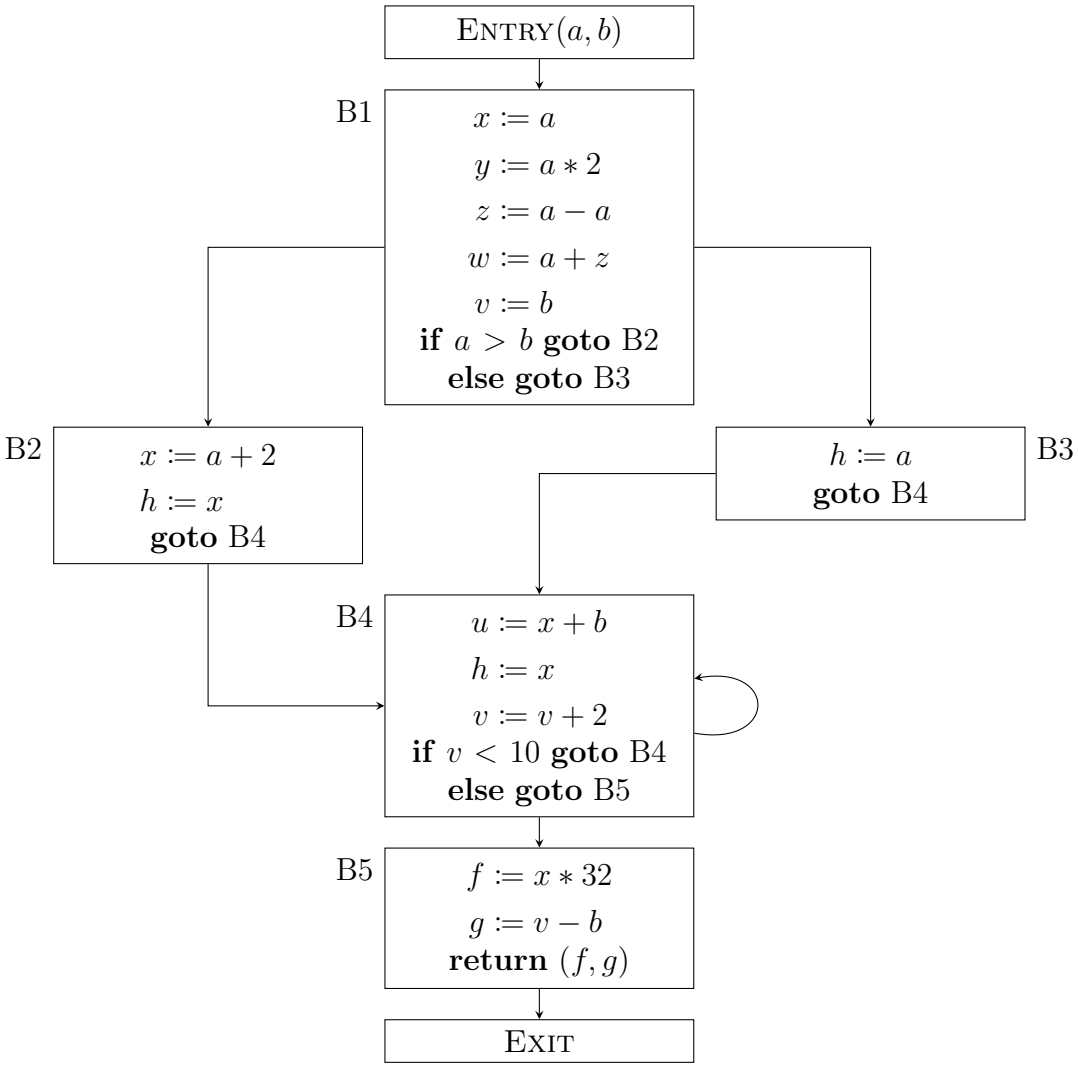
Solutions that correctly identify any mutated global state that affects execution of the program should receive full credit, provided they state the conditions that would lead to such state being modified and state why this needs to be maintained by the compiler. Solutions that do not state one or more of these should receive partial credit.

Note that in the absence of any global state, Simplicio's scheme is actually completely valid! This is similar to `setjmp`/`longjmp`, which is a common error handling scheme in C. It also resembles stack unwinding, but the unwind target can be determined statically as this is intraprocedural.

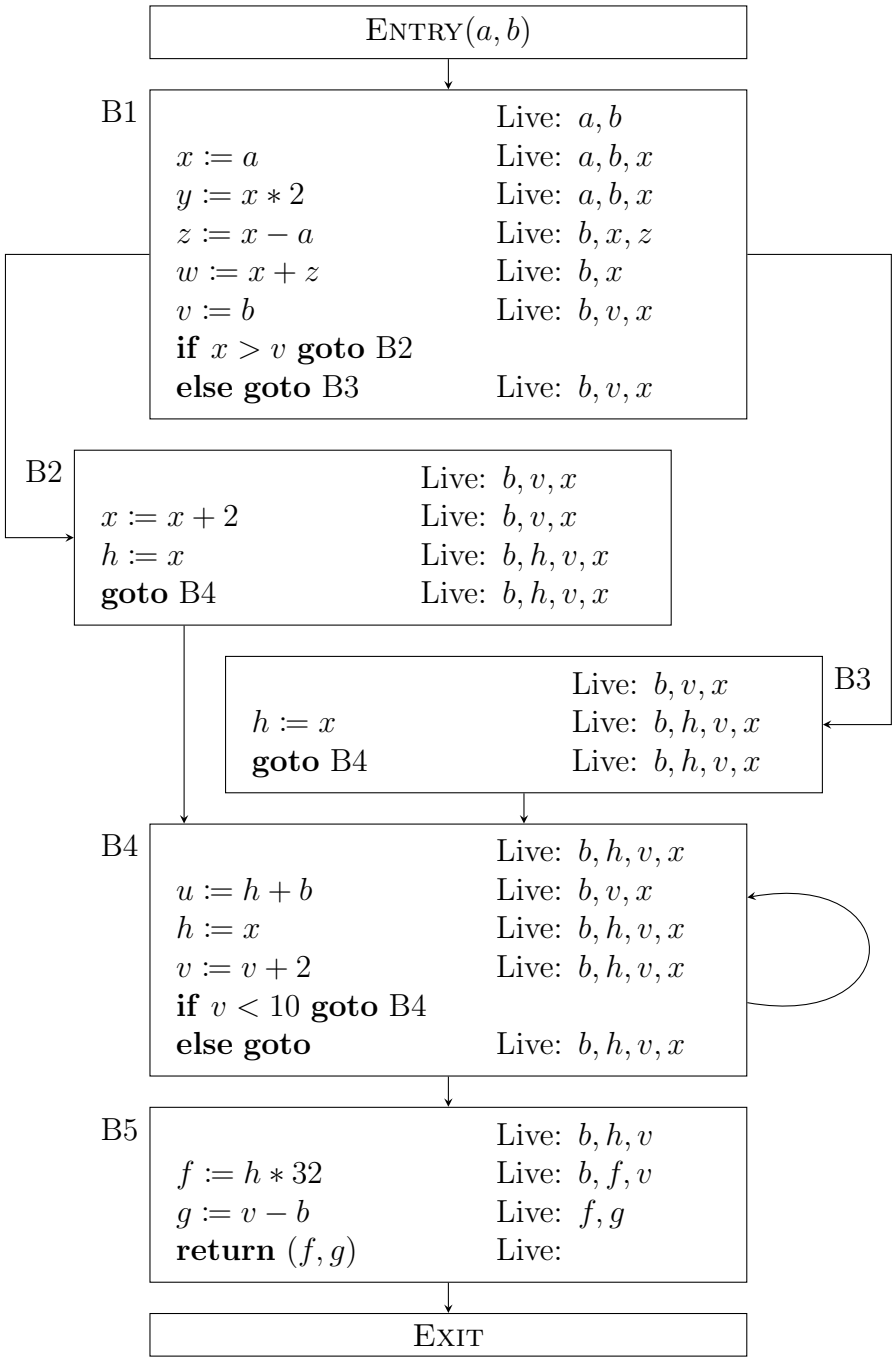
3. (a) Optimized CFG:



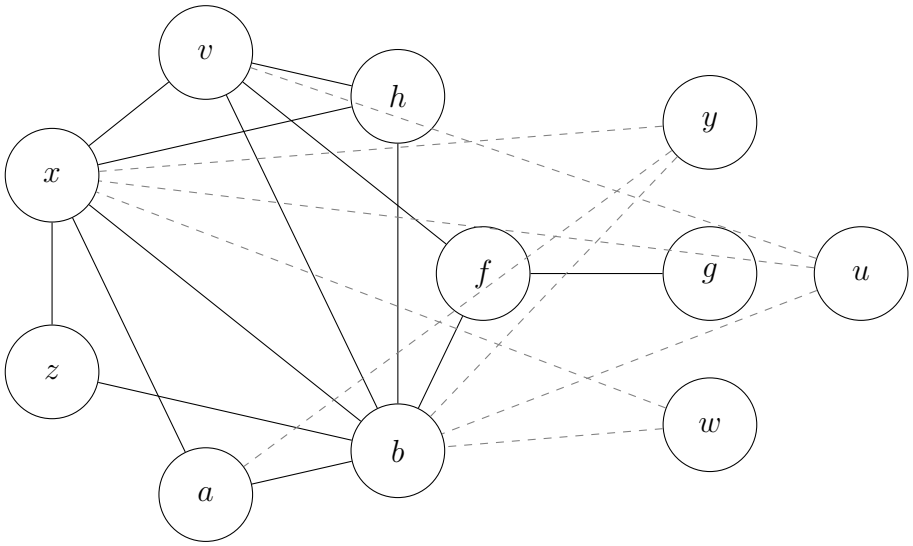
(b) Optimized CFG:



4. (a) Live variables:



(b) Interference graph:



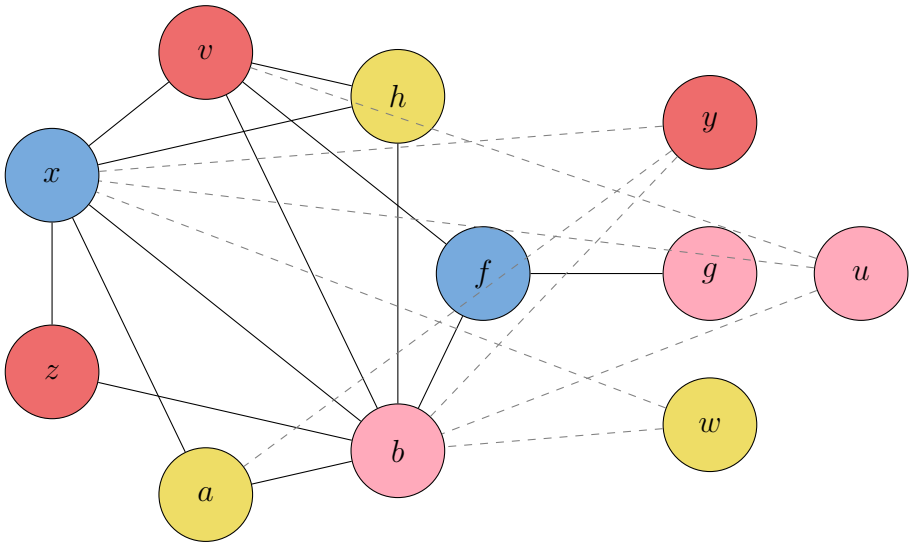

```
a - b;
a - x;
b - h;
b - v;
b - z;
b - x;
b - f;
f - g;
f - v;
h - v;
h - x;
v - x;
x - z;

extra:
a - y;
b - u;
b - w;
b - y;
u - v;
u - x;
w - x;
x - y;
```

The dashed edges arise from extending the live range of variables to include the point immediately after they are written. This is *not necessary* for full credit. To see why this is necessary, consider the following code:

```
1 add r1, r2          ; x += y
2 mov r1, something   ; z = something
3 add r2, r1          ; y += x
```

This arises by assigning registers $\mathsf{x} \mapsto \mathsf{r1}, \mathsf{y} \mapsto \mathsf{r2}, \mathsf{z} \mapsto \mathsf{r1}$. Note that this is not a valid assignment, as the assignment to `r1` overwrites the existing value of `x`, which is still live, even though `z` is not live. Dead code elimination can often eliminate such variables, but it is not valid to do so when instructions have side effects. For example, if `something` is a memory-mapped IO address, such as receiving data from an external device, then it would not be valid to remove the load.

(c) Coloring:



Any valid coloring is accepted for full credit.