CS143 Final
Spring 2026

- Please read all instructions (including these) carefully.
- There are 5 questions on the exam, some with multiple parts. You have 180 minutes to work on the exam.
- The exam is open note. You may use laptops, phones and e-readers to read electronic notes, but not for computation or access to the internet for any reason other than to access the class webpage.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. Do not write on the back of exam pages or other pages.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

SUNET ID: _____

NAME: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination. I have not used language models (LLMs).

SIGNATURE: _____

Problem	Max points	Points
1	25	
2	15	
3	15	
4	20	
5	25	
TOTAL	100	

1. Language Design

Suppose we extend Cool to include a for loop that counts up to a given number. The grammar gains the following production:

```
expr ::= ...
       | for ID <- e1 to e2 loop e3 pool
```

The loop behaves as follows: At the beginning of the loop, variable `ID` of type `Int` is defined and initialized to the value of `e1`. The expression `e2` is evaluated before each iteration of the loop and compared to `ID`. If `ID` is less than or equal to `e2`, then the body `e3` is evaluated followed by another loop iteration; otherwise the loop terminates. Finally, `ID` is incremented by 1 at the end of each iteration of the loop. The for loop expression evaluates to void.

For example, value of sum is 6 after the following loop has completed executing:

```
for i <- 0 to 3 loop sum <- sum + i pool
```

(a) Give the type checking rule of the `for` loop expression in the style of the Cool manual.

Answer:

The variable `ID` is bound with type `Int` in the body, exactly as **let** binds its variable; the whole expression has type `Object`:

$$\frac{O, M, C \vdash e_1 : \texttt{Int} \qquad O, M, C \vdash e_2 : \texttt{Int} \qquad O[\texttt{Int}/\texttt{ID}],\ M, C \vdash e_3 : T}{O, M, C \vdash \textbf{for}\ \texttt{ID} \leftarrow e_1\ \textbf{to}\ e_2\ \textbf{loop}\ e_3\ \textbf{pool} : \texttt{Object}}$$

(b) Give the operational semantics of the `for` loop expression in the style of Cool manual.

Answer:

We permit two solutions: `ID` may either be defined by the loop or defined before. The following solution defines it in the loop.

[Init case]:

$$\frac{\begin{array}{c} so, E, S \vdash e_1 : v_1,\ S_1 \\ l_{ID} = newloc(S_1) \\ E' = E[l_{\texttt{ID}}/\texttt{ID}] \\ S_2 = S_1[v_1/l_{\texttt{ID}}] \\ so, E', S_2 \vdash \textbf{body } \texttt{ID} \textbf{ to } e_2 \textbf{ loop } e_3 \textbf{ pool} \end{array}}{so, E, S \vdash \textbf{for } \texttt{ID} \leftarrow e_1 \textbf{ to } e_2 \textbf{ loop } e_3 \textbf{ pool} : void, S_2}$$

[Iteration case]:

$$\frac{\begin{array}{c} so, S, E \vdash e_2 : v_{bound}, S_1 \\ E(ID) = l_{ID} \\ S_1(l_{ID}) = v_{ID} \\ v_{ID} \leq v_{bound} \\ so, E, S_1 \vdash e_3 : v, S_2 \\ S_3 = S_2[v_{\texttt{ID}} + 1/l_{\texttt{ID}}] \\ so, E, S_3 \vdash \textbf{body } \texttt{ID} \textbf{ to } e_2 \textbf{ loop } e_3 \textbf{ pool} : void, S_4 \end{array}}{so, E, S \vdash \textbf{body } \texttt{ID} \textbf{ to } e_2 \textbf{ loop } e_3 \textbf{ pool} : void,\ S_4}$$

[Termination case]:

$$\frac{\begin{array}{c} so, S, E \vdash e_2 : v_{bound}, S_1 \\ E(ID) = l_{ID} \\ S_1(l_{ID}) = v_{ID} \\ v_{ID} > v_{bound} \end{array}}{so, E, S \vdash \textbf{body } \texttt{ID} \textbf{ to } e_2 \textbf{ loop } e_3 \textbf{ pool} : void,\ S_1}$$

(c) Write the MIPS code generation function of the `for` loop expression. You may use the same type of pseudo code as in the lecture notes (e.g., Lecture 12, Slide 13), but do not need to use different colors to distinguish between compiler code and generated code.

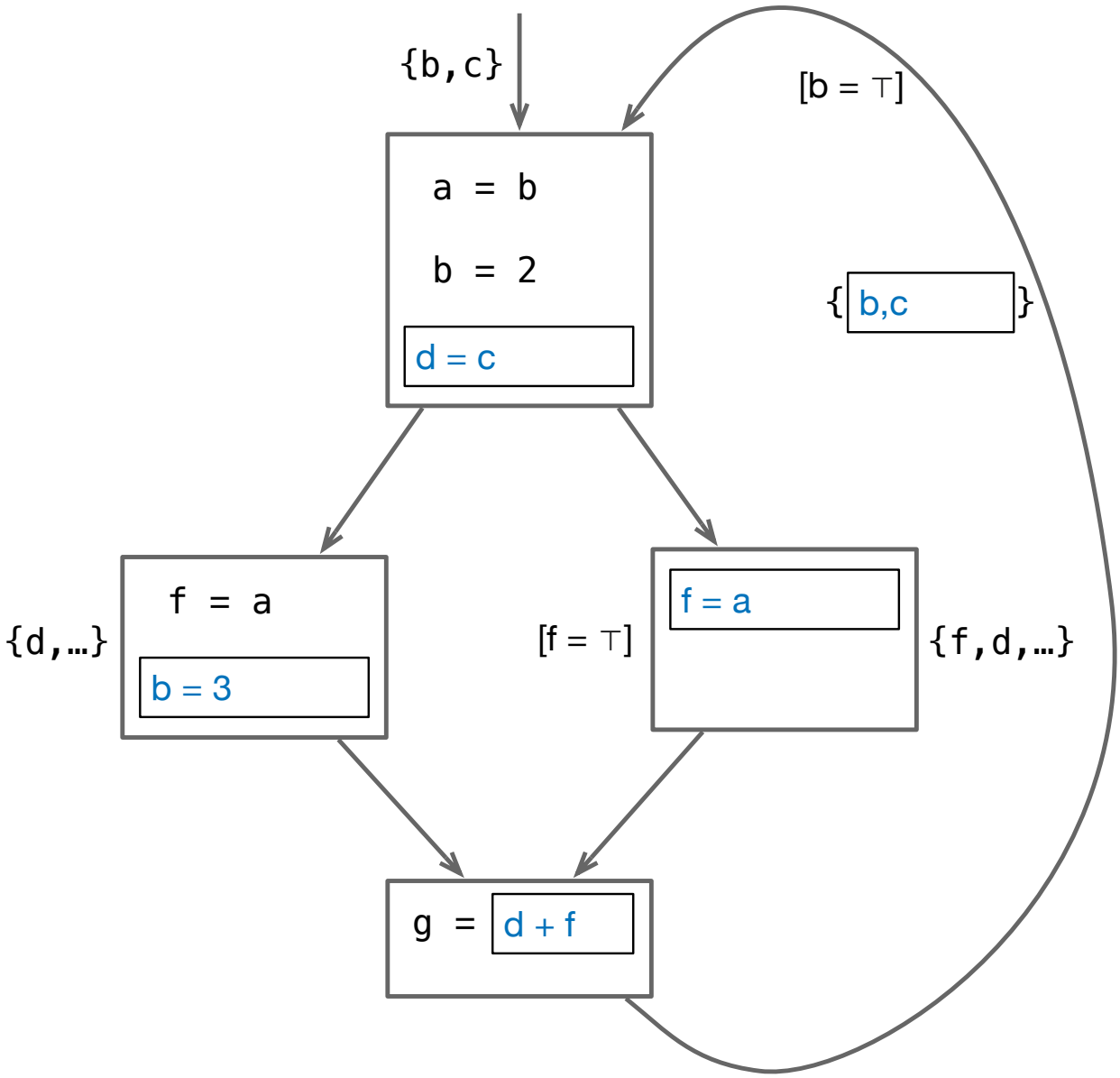
Answer:

```
cgen(for ID <- e1 to e2 loop e3 pool) =
    cgen(e1)                    # start value -> $a0
    sw    $a0 0($sp)
    addiu $sp $sp -4
    loop_top:
        cgen(e2)                # loop bound -> $a0
        lw    $t1 4($sp)        # load ID
        bgt   $t1 $a0 loop_end  # exit if ID > bound
        cgen(e3)                # evaluate body (result discarded)
        lw    $a0 4($sp)        # load ID
        addiu $a0 $a0 1         # ID + 1
        sw    $a0 4($sp)        # store incremented ID
        b     loop_top
    loop_end:
        addiu $sp $sp 4         # pop both temporaries
```

2. Dataflow Analysis

The following control-flow graph consists of statements inside the basic block, partial liveness information in curly brackets (e.g., `{d,...}`), and partial constant propagation information in square brackets (e.g., `[b=⊤]`). Only `b` and `c` are live on entry. Please fill in the missing information where there are empty rectangles. The missing information is the last statement of the top three basic block, the missing right-hand side of the statement in the bottom basic block, and the liveness information on the back-edge. Statements may assign a variable/constant to a variable (e.g., `a = b`) or add variables (e.g., `a = b + c`).

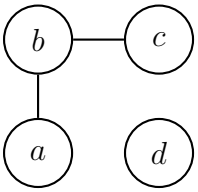
Answer:



The right block can also be `f = d`, or `f = 2 + 3`. The left block can assign `a` or `d` to `b`.

3. Register Allocation

Give the shortest possible program that results in the following register inference graph:



Your program must satisfy the following conditions:

- All statements must be of the form `a=b+c` where `a`, `b`, and `c` are program variables (note, they need not be literally just `a`, `b`, and `c` but may be any variable name).
- Only the variable on the left-hand side of the last is live on exit from the program.

Do not be concerned with how variables are defined/initialized: any number of variables may be live on entry to the program. Explain why your solution is the shortest possible program.

Answer:

```
a = b + c
d = a + b
```

or

```
c = a + b
d = b + c
```

At least two statements are needed to introduce four variables, since the problem specification states that only the left-hand side of the last statement is live on exit.

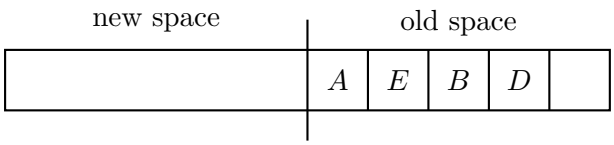
4. Garbage Collection

In the following questions, you will be asked to write code that results in various heap states given different garbage collection algorithms. Your answers should be short programs consisting of sequences of statements separated by newlines. The statements may allocate objects with `new` expressions, read/write to local variables, and read/write to attributes. There is no need to provide class types and you may assume objects have as many attributes as you need using the naming convention `a`, `b`, `c`, and so forth. For example:

```
a   = new  // Allocates an object
a.f = a    // Sets a.f to point to the same object as a
a = null   // Set a reference to point to null
```

You may allocate objects with a `new` expression, assign to local variables, and assign to attributes.

(a) The following diagram shows the state of the heap right after stop-and-copy has reclaimed memory. Arrows showing references between objects are intentionally left out.



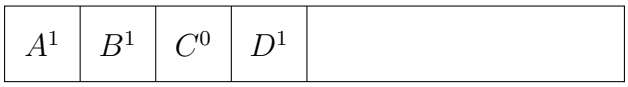
Write statements that, after stop-and-copy garbage collection, results in the above heap state. The program must allocate *five* objects named `A`, `B`, `C`, `D`, and `E`, *in that order*. That is, the first call to `new` allocates `A`, the second allocates `B`, and so forth. During garbage collection, attributes are traversed in alphabetical order (`a`, `b`, ...). At the end of your statements, only one local variable should be pointing to an object.

Answer:

Several solutions are possible, but here's one.

```
a     = new  // object A
a.b   = new  // object B
a.b.a = new  // object C
a.b.a = new  // object D
a.a   = new  // object E
```

(b) The following diagram shows the state of the heap right after the mark phase of mark-and-sweep. The superscripts show the state of the mark bit (0 or 1).



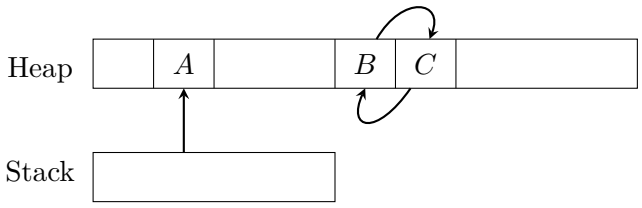
Write statements that results in this heap state.

Answer:

Several solutions are possible, but here's one. Since C's mark bit is zero, it means it is not reachable, so we have to ensure no reachable variable or attribute is pointing at it.

```
a = new  // object A
b = new  // object B
d = new  // object C
d = new  // object D
```

(c) Give a sequence of statements that results in the following heap state if reference counting is used for garbage collection. Assume all local variables live on the stack.



Answer:

Several solutions are possible, but here's one. The key is to set up a cycle between `b` and `c` that is unreachable from local variables.

```
a   = new  // object A
b   = new  // object B
c   = new  // object C
b.a = c
c.a = b
b = null
c = null
```

5. Code Generation and Runtime Organization

Below is the partial output of a Cool compiler. Write Cool source code that could compile to this assembly. You only need to show the fragment corresponding to this code (and do not need to show the complete Cool program.

```
int_const0:
  .word 2
  .word 4
  .word Int_dispTab
  .word 0
  .word -1

bool_const0:
  .word 3
  .word 4
  .word Bool_dispTab
  .word 0
  .word -1

bool_const1:
  .word 3
  .word 4
  .word Bool_dispTab
  .word 1
```

Answer:

```
Class Main {

  a : Int;

  f(x: Int, y: Int): Int {
    if 0 < x then
      a <- a + y
    else
      0
    fi
  };
  ...
};
```

```
Main.f:
  addiu  $sp $sp -20
  sw     $fp 12($sp)
  sw     $s0 8($sp)
  sw     $ra 4($sp)
  addiu  $fp $sp 16
  move   $s0 $a0
  sw     $s1 4($fp)
  la     $s1 int_const0
  lw     $a0 12($fp)
  lw     $t1 12($s1)
  lw     $t2 12($a0)
  la     $a0 bool_const1
  blt    $t1 $t2 label2
  la     $a0 bool_const0
label2:
  lw     $t1 12($a0)
  beqz   $t1 label0
  lw     $s1 12($s0)
  lw     $a0 8($fp)
  jal    Object.copy
  lw     $t2 12($a0)
  lw     $t1 12($s1)
  add    $t1 $t1 $t2
  sw     $t1 12($a0)
  sw     $a0 12($s0)
  b      label1
label0:
  la     $a0 int_const0
label1:
  lw     $s1 4($fp)
  lw     $fp 12($sp)
  lw     $s0 8($sp)
  lw     $ra 4($sp)
  addiu  $sp $sp 28
  jr     $ra
```

(blank page for extended answers)