Student Name: _____
Student Number: _____
School: _____
Year of Entrance: _____

ShanghaiTech University final exam Cover Sheet

Academic Year : 2022 to 2023 Term: 2
Teaching School: School of Information Science and Technology
Instructor: songfu
Course Name: Compilers
Course Number: CS131.01

Exam Instructions for Students:

1. All examination rules must be strictly observed during the entire exam, and any form of cheating is prohibited.

2. Other than allowable materials, students taking closed-book tests must place their books, notes, tablets and any other electronic devices in places designated by the examiners.

3.Students taking open-book tests may use allowable materials authorized by the examiners. They must complete the exam independently without discussion with each other or exchange of materials.

For Marker's Use:

Section	1	2	3	4	5	6	7	8	9	10	Total
Marks											
Recheck											

Marker's Signature: **Reviewer's Signature:**
Date: **Date:**

Instructions for Examiners:

1. The format of the exam papers and answer sheets shall be determined by the school and examiners according to actual needs. All pages should be marked by the page numbers in order (except the cover page). All text should be legible, visually comfortable and easy to bind on the left side. A4 double-sided printing is recommended for the convenience of archiving (There are all-in-one printers in the university).

2. The examiners should make sure that exam questions are accurate and appropriate. If students have any enquiries about the exam questions during the exam, the examiners should be responsible to respond on site, which will be taking into account in the teaching evaluation.

Question:	1	2	3	4	5	Total
Points:	15	27	22	20	16	100
Score:						

1. (15 points) **(Code Optimization)** Consider the following piece of python code. Perform the optimization directly on the code until they can not be optimized.

```
def func(x)
    p = 3
    r = 10
    s = p + r
    t = 2 * r + s
    t = p
    u = p + r
    v = p + t
    w = 3 + x
    return u,v,w
```

Figure 1: Oringal Code

Hint: Typical optimization approaches are Algebraic Simplification, Copy Propagation, Constant Folding, Dead Code Elimination and Common Sub-expression Elimination. **The process is required**.

2. **(Data Flow Analysis)** Read the materials and answer the questions.
   (a) (5 points) Explain the meaning of **Reaching Definition**.

   (b) Read the following data flow graph, and answer the following questions.

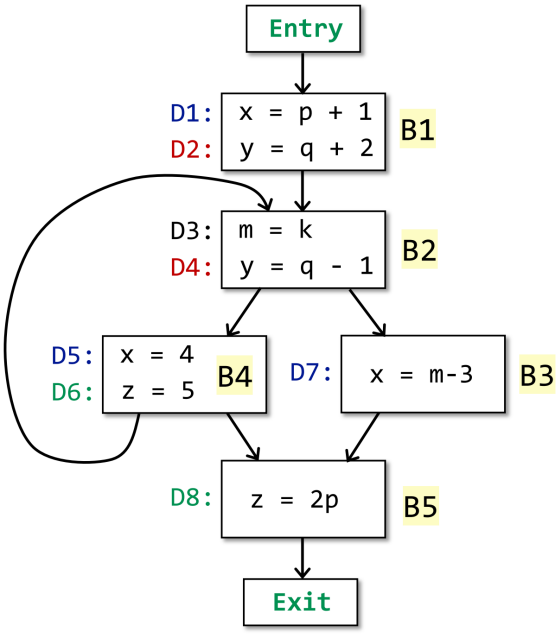


Figure 2: control flow graph

   i. (10 points) The Reaching Definitions at Exit stage.

ii. (12 points) Show your iteration process finding the Reaching Definition at Exit stage. A copy of the figure is here for your convenience.

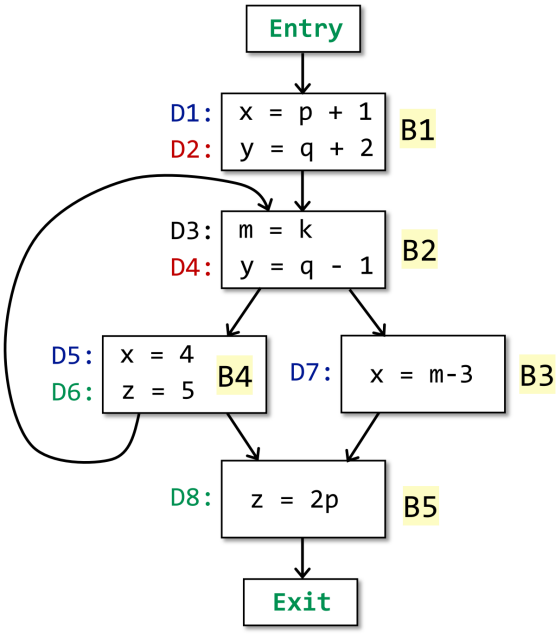
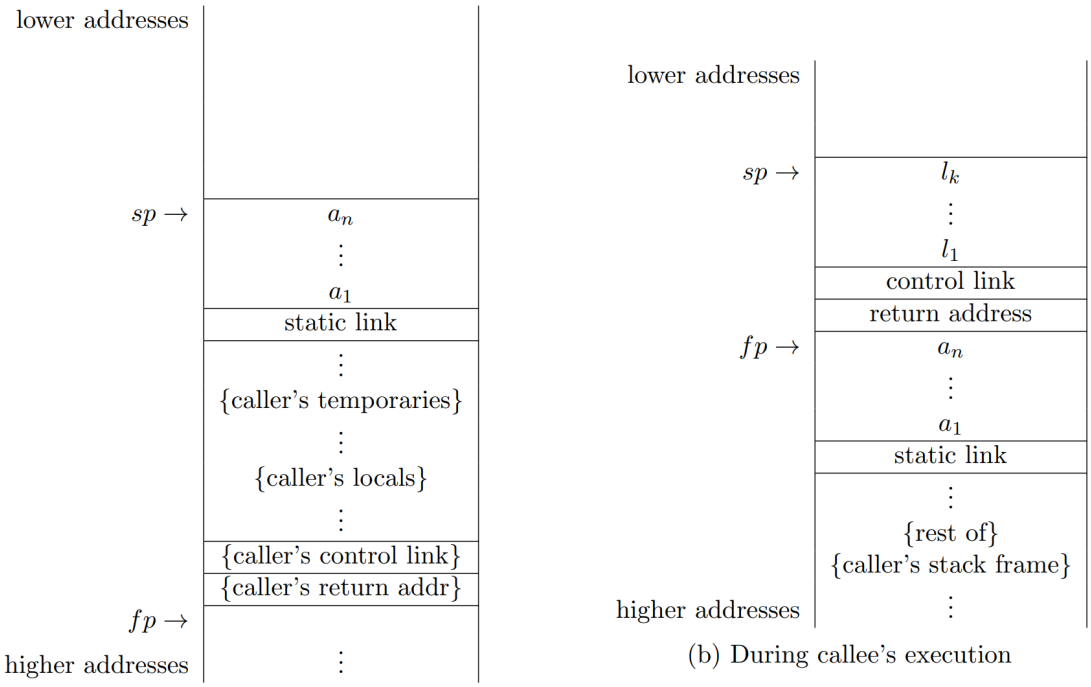


Figure 3: control flow graph

3. **(Runtime Environment)** In the following ChocoPy code, we are computing a binomial number recursively. Refer to the stack layout of the ChocoPy.

```
def C(n:int,m:int) -> int:
    a:int = 0
    b:int = 0
    if(n<m || m<0):
        return 0
    if(m==0):
        return 1
    a = C(n-1,m)
    b = C(n-1,m-1)
    return a + b
```



(a) Before/after invocation

(b) During callee's execution

Stack layouts during the invocation of a nested function with n arguments $a_1, \ldots, a_n$ and k local variables having initial values $l_1, \ldots, l_k$. The diagram on the left shows the state of the stack just before the function is invoked (and just after it returns). The diagram on the right shows the state of the stack just before executing the called function's first statement. The stack layouts are similar during invocations of global functions and methods, except that the static link is not present.

Figure 4: ChocoPy Stack Layout

The questions are on the following pages:

(a) (12 points) Suppose we are calling C(4,2), draw the full activation tree.

(b) (10 points) Suppose we are calling C(4,2), show the stack configuration when the first C(2,2) is about to return.

4. **(Data flow Analysis and Register Allocation)** Consider the following program:

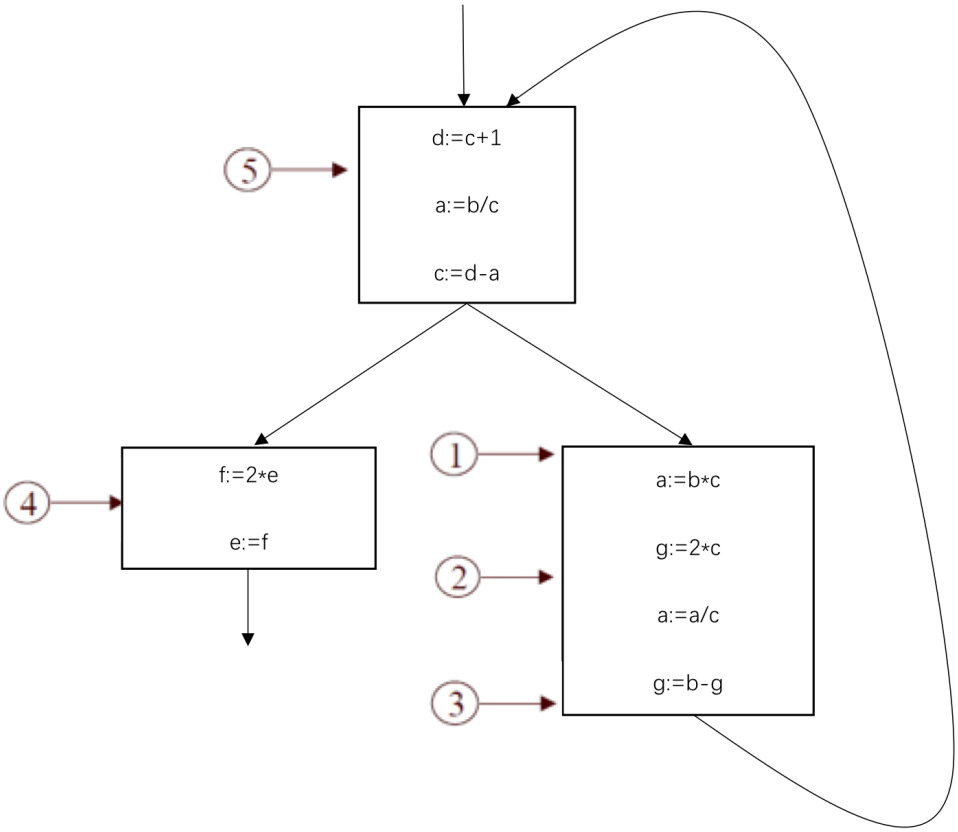


Figure 5: Data Flow Graph of 4.

(a) (8 points) Compute the set of live variables at the points 1-5 in the program. Assume that there are no live variables on exit. The process **is not required**.

(b) (6 points) Refer to Figure 6, draw the register interference graph of the program (you should use the same topology).

(c) (6 points) In Figure 6, show a coloring for register interference graph using the minimal number of colors. Using color names such as R1, R2, R3. Write down the color next to each node in your graph. Make your response look smart.

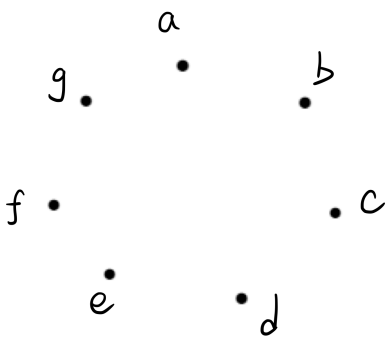


Figure 6: The interference graph

5. **(Garbage Collection)** Answer the following questions

   (a) (3 points) What's the advantage and disadvantage of Mark-and-Sweep.

   (b) (3 points) What's the advantage and disadvantage of Stop-and-copy.

   (c) (3 points) What's the advantage and disadvantage of Reference-Counting.

   (d) (7 points) Your friend Long wants to mitigate the runtime overhead of automatic memory management by combining static and dynamic techniques. In particular, he would like his compiler to automatically insert as many delete calls as possible and have only the remaining objects collected at run-time by the garbage collector. After reading over your CS131 lecture notes, he decides liveness analysis is a great match for this problem. Specifically, he proposes to insert a call **delete(v)** to delete the object referred by v as soon as the variable **v** becomes dead. Is this a correct memory management strategy? Explain your reasoning.