CS614: Advanced Compilers

Mid-Sem Exam (2 hours; 20 marks)

September 21st, 2023

Instructions:

- Write neat, clear and crisp answers.
- In case you make an assumption, write it down before the corresponding answer.
- Each **Q** carries 5 marks, along with a 1 mark bonus at the end.

Q1: FIRST UNDERSTAND THE BITS AND BYTES

(A) Match each error with the phase that usually detects the same: [1]

(a) Included file does not exist	(i) Semantic analysis
(b) Array out of bounds	(ii) Syntax analysis
(c) Semicolon missing	(iii) Preprocessing
(d) Undeclared variable	(iv) Run-time

For questions (B) – (E), consider the following piece of Java Bytecode:

```
1     bipush 5
2     istore_1
3     bipush 20
4     istore_2
5     iload_2
6     iload_1
7     isub
8     istore_3
9     iload_3
10    ireturn
```

(B) Assuming explicit operands take a byte, what should be the size of the activation record corresponding to a method containing only the above piece of Bytecode? Assume the method does not take any parameters. [1]

(C) Write a Java code snippet corresponding to the above Bytecode sequence. [1]

(D) Apply constant propagation and constant folding to optimize the Java code that you have written, while showing the resultant code after each optimization, in sequence. [1]

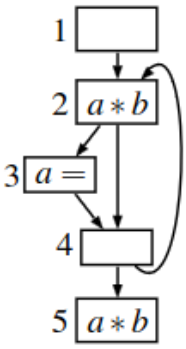
(E) Rewrite the Bytecode corresponding to the optimized Java code and compute the percentage memory saving over the unoptimized version [1].

Q2: START PARTIALLY AND FINISH GLOBALLY

(A) Perform partial redundancy elimination (PRE) for the following program. Give the result in the following format. Instantiate the column of Iteration i as many times as needed. [3]

Node	First level properties				Initialization		Iteration i		*Insert*	*Redundant*
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*		

The required dataflow equations are as follows:



$$PavIn_n = \begin{cases} \emptyset & n \text{ is } Start \text{ block} \\ \bigcup_{p \in pred(n)} PavOut_p & \text{otherwise} \end{cases}$$

$$PavOut_n = (PavIn_n - Kill_n) \cup Gen_n$$

$$AvIn_n = \begin{cases} \emptyset & n \text{ is } Start \text{ block} \\ \bigcap_{p \in pred(n)} AvOut_p & \text{otherwise} \end{cases}$$

$$AvOut_n = (AvIn_n - Kill_n) \cup Gen_n$$

$$In_n = PavIn_n \cap (AntGen_n \cup (Out_n - Kill_n)) \bigcap_{p \in pred(n)} (Out_p \cup AvOut_p)$$

$$Out_n = \begin{cases} \emptyset & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

(B) Differentiate PRE and global value numbering (GVN). Can you come up with a mechanism to perform PRE and GVN in a way that one could take advantage of the other? [2]

Q3: CONSTRUCT AND DESTRUCT INTERMEDIATE SOLUTIONS

(A) For the following program snippet, draw a CFG (with designated entry and exit nodes) and then convert the same to SSA form. In the plain CFG version show additionally the def-use edges (draw dashed lines). In the SSA version show additionally the SSA edges (dashed lines). You can assume that all the variables are declared globally. [2]

```
1    void foo() {
2        k = false;
3        i = 1;
4        j = 2;
5        while (i <= n) {
6            j = j * 2;
7            k = true;
8            i = i + 1;
9        }
10       if (k) {
11           print j;
12       } else {
13           i = i + 1;
14       }
15   }
```

(B) Recall that SSA is just an IR and that a compiler usually needs to "destruct" the same before code generation. Say Teja proposed the following algorithm for SSA destruction: *Insert a copy statement at the end of each predecessor block of the ϕ-node, corresponding to each ϕ-node's argument.* An example of Teja's algorithm is shown in Figure 1; the code in Figure 1b is the SSA-destructed version of the code in Figure 1a.

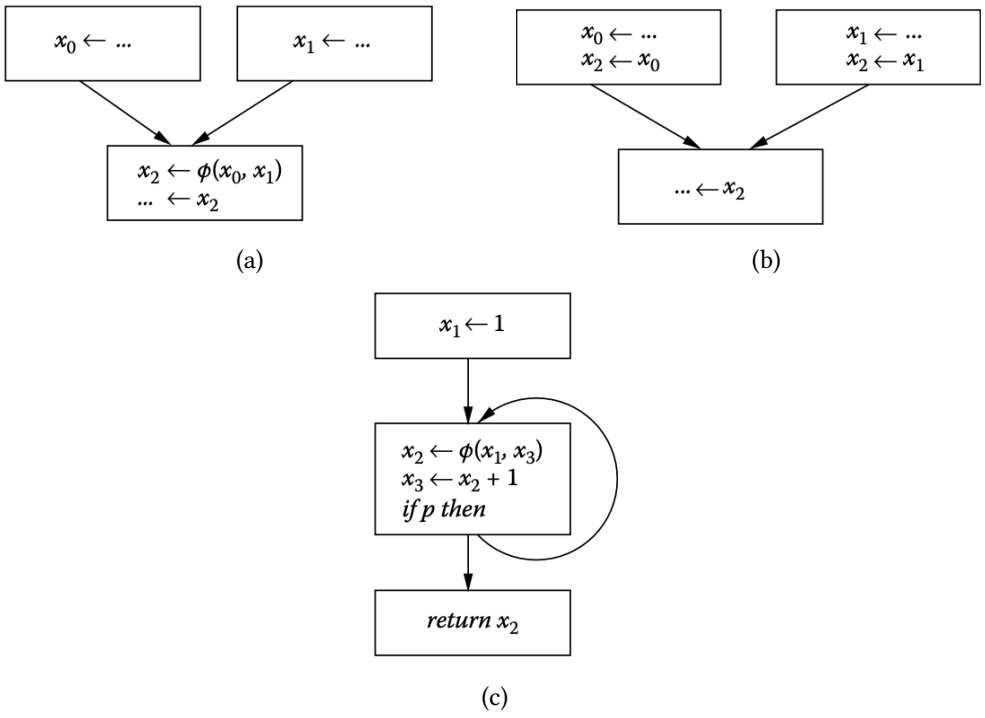


Figure 1: Q2 accompaniments.

Swetha claims that the algorithm leads to an issue while SSA-destructing the program in Figure 1c.

(i) First apply Teja's algorithm on the code in Figure 1c and show that Swetha's problem holds true. [1]

(ii) Now propose a modification to Teja's algorithm (we can name it on you) that solves the above problem, along with the result of applying your algorithm on the code in Figure 1c. [2]

Q4: PUT CONNECTIONS AND YOU FINALLY GET ADVANCED STRUCTURES

Recall the way fields and methods are laid out in Java (the jol examples and vtables). Assume the following number of bytes for different types of values:

Type	Size (in Bytes)
references	4
booleans	1
bytes	1
chars	2
ints	4
longs	8

(A) Draw the object structure for the two classes below, with the pre Java8 strategy (declaration order followed for fields). Explictly indicate how many bytes would class `C` and `D` objects require, and the total amount of internal and external padding for each of them. [2]

```
1    class C {
2        boolean firstField;
3        C secondField;
4        void methodOne() { ... }
5    }
```

```
6    class D extends C {
7        boolean firstField;
8        long secondField;
9        char thirdField;
10       int fourthField;
11       void methodOne() { ... }
12       void methodTwo() { ... }
13   }
```

(B) Draw the object structure for class `D`, with the Java8+ strategy (minimize internal padding). [1]

(C) In order to ensure that two particular objects are loaded in two different cache blocks – solution to a problem called *false sharing* – many C/C++ programmers tend to insert dummy fields into a structure/class. An example is given below; `helpMe` is the field being tried to be separated across cache blocks by inserting all the `long` fields (assuming each cache block is ≤ 64 bytes). Explain why this solution might not work in Java. Can you suggest how could the class structure be modified to solve false sharing even in this example? (You can ask the instructor for a small hint, either by redeeming a PC or at the cost of half a mark.) [2]

```
1    public class LongPadding {
2        long l01, l02, l03, l04, l05, l06, l07, l08; // 64 bytes
3        byte helpMe;
4        long l11, l12, l13, l14, l15, l16, l17, l18; // 64 bytes
5    }
```

WELL, SOMETIMES THERE IS BONUS WORK

Go to the following link, fill a short course feedback, and *then* end your answer sheet with a "Done". `https://bit.ly/cs614-a23-msf` [1]

--*-* HO GAYA SAMAAPT *-*-*-*

"An exam's purpose is not only to test what was learnt in the class, but to learn what was missed in the class."