SWE 300-01
Spring 2020
Final
May 7
Time Limit: 1440 Minutes

Name: Andrew Januszko

This exam contains 7 pages (including this cover page) and 4 questions. Total of points is 60. There are 0 bonus points.

The exam is individual. The exam is open-book and open-notes. You cannot use the internet.

Please submit the exam on Gradescope by 11:59p May 7.

Grade Table (for instructor use only)

Question	Points	Bonus Points	Score
Refactoring & tuning	17	0	
Concurrency	12	0	
Array Allocation	11	0	
Garbage Collection	20	0	
Total:	60	0	

1. (a) (3 points) Jam the loops:

```
int a[10], b[10];
for (int i=0; i<10; i++) {
  a[i] = i+2;
}
for (int i=0; i<10; i++) {
  b[i] = i*2;
}
```

```
int a[10], b[10];
for (int i = 0; i < 10; i++) {
  a[i] = i + 2;
  b[i] = i * 2;
}
```

(b) (2 points) Explain why loop jamming may speed up the code.

It speeds up the code because it does not have to take the time to do another for loop unlike the original

(c) Consider the following code:

```
public class Fib {
  static int fib(int n) {
    if (n == 0 || n == 1 || n == 2) return 1;
    return fib(n-1) + fib(n-2);
  }
}
```

i. (4 points) Write down the results of fib(1), fib(2), fib(3), fib(4), and fib(5).

fib(1) = 1, fib(2) = 1, fib(3) = 2, fib(4) = 3, fib(5) = 5

ii. (4 points) What will happen if the argument to fib is negative? How do you make the code "defensive"? Show changed code in the following box.

If fib is negative, then the program will be stuck in an infinite loop.

```
public class Fib {
  static int fib(int n) {
    if (n < 0) return 0;
    if (n == 0 || n == 1 || n == 2) return 1;
    return fib(n-1) + fib(n-2)
  }
}
```

iii. (4 points) When computing fib(5) = fib(4) + fib(3), you might notice that fib(3) will be computed more than once since fib(4) = fib(3) + fib(2). Change the original Fib so that it leverages memoization.Assume n is less than 16. Hint: Use an extra array.

```
public class Fib {
  int fibMem[16];
  static int fib(int n) {
    if (n == 0 || n == 1 || n == 2) return 1;
    if (n < 0) return 0;
    if (fibMem[n] =/= 0) {
      return fibMem[n];
    } else {
      fibMem[n] = fib(n-1) + fib(n-2);
      return fibMem[n];
    }
  }
}
```

2. (a) (2 points) Describe how the dead lock happened in the dining philosopher example.

The deadlock occurs when everyone picks up their left fork at once, leaving no available forks for the right.

(b) (8 points) Adam and Betty share a bank account. The pseudo code (looking suspiciously like Java though) of the bank system is listed below:

```
1 class Bank {
2   int balance;
3   //...
4   void withDraw(int amount) {
5     int oldBalance = balance;
6     int newBalance = oldBalance - amount;
7     balance = newBalance;
8   }
9 }
```

Assume each statement is *atomic* but the method `withDraw` is not. For example, you can assume line 6 is atomic and will not be interrupted. The starting balance of the account is $100. If Adam withdraws $45 and Betty withdraws $30, the balance may not be the expected $25.

Demonstrate how *the balance* can be either $70 and $55 in the following two tables. Only one statement can be executed in each row. Write down the value of `balance` in the first column *before* executing the statement in the current row.

Demonstrate that the balance could end up $55. You can choose just to write down the line number.

Balance	Adam	Betty
100	oldBalance = 100	oldBalance = 100
100	oldBalance = 100	
100		newBalance = 100 - 30 = 70
100	oldBalance = 100	
100	newBalance = 100 - 45 = 55	
100		balance = 70
55	balance = 55	

Demonstrate that the balance could end up $70. You can choose just to write down the line number.

Balance	Adam	Betty
100	oldBalance = 100	oldBalance = 100
100		oldBalance = 100
100	newBalance = 100 - 45 = 55	
100		oldBalance = 100
100		newBalance = 100 - 30 = 70
100	balance = 55	
70		balance = 70

(c) (2 points) Help the bank prevent this from happening. Show the improved `Bank` in the following box.

```
class Bank {
  int balance;
  // ...
  void withDraw(int amount) {
    synchronized (balance) {
      int oldBalance = balance;
      int newBalance = oldBalance - amount;
      balance = newBalance;
    }
  }
}
```

3. In this section, you will implement an array list.

```
public class IntList {
  private static int DEFAULT = 16;
  private int[] arr;
  private int size = 0; /* size is used to track the number of actual elements */
  private int capacity; /* How will use this variable? */
  public IntList() { /* Initialize the underlying array to the default size */ }
  public void add(int n) { /* Add n to the end of the list */ }
  public void remove() { /* Remove the last element of the list */ }
}
```

If you don't like Java, you can choose to implement the alternative C version.

```
#define DEFAULT 16
```

```c
typedef struct {
  int arr[DEFAULT]; int size; int capacity;
} IntList;

IntList *IntList() { /* ... */ }
void add(IntList *lst, int n) { /* ... */ }
void remove(IntList *lst) { /* ... */ }
```

(a) (2 points) Implement the constructor `IntList`.

```
public IntList() {
    this.capacity = DEFAULT;
    arr = new int[DEFAULT];
  }
```

(b) (4 points) Implement `add`. On top of just adding an element, at some point we may have more elements than the default size and that's when we will expand the underlying array `arr`. Use the strategy described in the course.

```
public void add(int n) {
    if (size == capacity) {
      arr = Arrays.copyOf(arr, size + 1);
      capacity = capacity + 1;
    }
    arr[size] = n;
    size = size + 1;
  }
```

(c) (5 points) Implement `remove`. On top of just removing an element, at some point we may have an underlying array that is too large and too few elements. We should shrink the underlying array `arr`. Use the strategy described in the course.

```
public void remove() {
    if (size > 0) {
      arr = Arrays.copyOf(arr, size - 1);
      size = size - 1;
      capacity = capacity - 1;
    }
  }
```

4. (_____/*20 points*) Garbage collection:

(a) (4 points) What are *syntactic garbage* and *semantic garbage*?

Syntactic garbage is the memory impossible to reach, semantic garbage is the memory in fact will never be used (freeing a chunk of memory).

(b) Assume a machine only has 8 memory cells. Each memory cell can contain either a pointer `px` when `x` is the destination the pointer points to or a number `nx` where `x` is the actual number. The current memory content is shown below:

1	2	3	4	5	6	7	8
p3	n2	n7	p1	p7	n12	p3	n6

i. (4 points) Which memory cells will be marked as unreachable in the mark stage? Demonstrate how you find them.

Cells 2, 6, and 8 are marked for deletion.

1:p3 → 3:n7
4:p1 → 1:p3 → 3:n7
5:p7 → 7:p3 → 3:n7
7:p3 → 3:n7

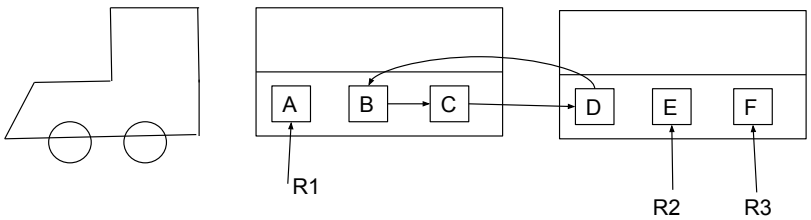
ii. (4 points) What are the advantage and disadvantage if reachable memory cells are moved together in the sweep stage?
Advantage:

Memory cells would be all together leading to no fragmentation.

Disadvantage:

The program would freeze for a longer period of time while it defragments the memory cells.

(c) Train algorithm. The figure below illustrates the memory management situation in shapes of trains.



i. (2 points) Write down the remembered sets of the train and the cars in the figure above.

A,E,F

ii. (6 points) Show step by step how B,C,D are collected.

```
Step #1
Delete D in car #2
move A to car #2
making the train → [B|C][A|E|F]

Step #2
Delete C in car #1
making the train → [B][A|E|F]

Step #3
Delete B in car #1
making the train → [A|E|F]

Step #4
Delete car #1
making the train → [A|E|F]
```