

ESOF 322 Software Engineering I
Homework #5 (32 points total)

Due: Tuesday November 5th

Hand in your printed copy in class, and submit a PDF file into D2L.

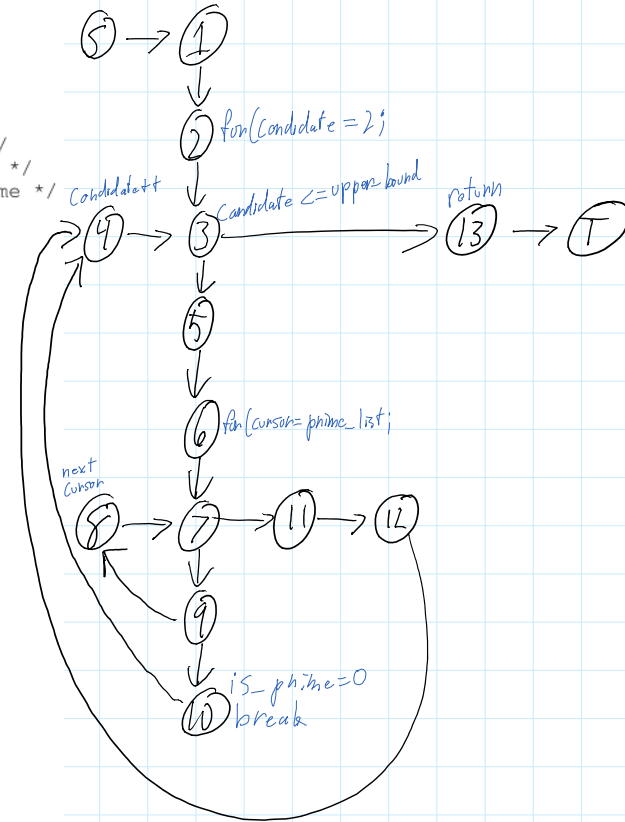
Question 1 (10 pts)

- a) Create a control flowgraph for the *sieve* algorithm. To the left of the line numbers in the source code clearly identify the nodes that will be used in your graph. Once you have identified the nodes, draw the control graph. (4 pts)

```

1. /* Find all primes from 2-upper_bound using Sieve of Eratosthanes */
2.
3. #include
4. typedef struct IntList {
5.     int value;
6.     struct IntList *next;
7. } *INTLIST, INTCELL;
8. INTLIST sieve ( int upper_bound ) {
9.
10.     INTLIST prime_list = NULL; /* list of primes found */
11.     INTLIST cursor; /* cursor into prime list */
12.     int candidate; /* a candidate prime number */
13.     int is_prime; /* flag: 1=prime, 0=not prime */
14.
15.     /* try all numbers up to upper_bound */
16.     for (candidate=2;
17.
18.3- candidate <= upper_bound;
19. candidate++)
20.
21.5- is_prime = 1; /* assume candidate is prime */
22. for(cursor = prime_list;
23.
24. cursor;
25.8- cursor = cursor->next)
26.
27. if (candidate % cursor->value == 0)
28.
29.     /* candidate divisible by prime */
30.     /* in list, can't be prime */
31.     is_prime = 0;
32.     break; /* "for cursor" loop */
33.
34.
35. if(is_prime)
36.
37.     /* add candidate to front of list */
38.     cursor = (INTLIST) malloc(sizeof(INTCELL));
39.     cursor->value = candidate;
40.     cursor->next = prime_list;
41.     prime_list = cursor;
42.
43.
44. return prime_list;
45. }

```



- b) Provide a set of test cases that would give 100% Node Coverage (NC). (2 pts)

{4} (i.e. execution of sieve(4)) covers 100% of nodes in sieve()

- c) Provide a set of test cases that would give 100% Edge Coverage (EC). (2 pts)

{4} (i.e. execution of sieve(4)) covers 100% of edges in the graph of sieve()

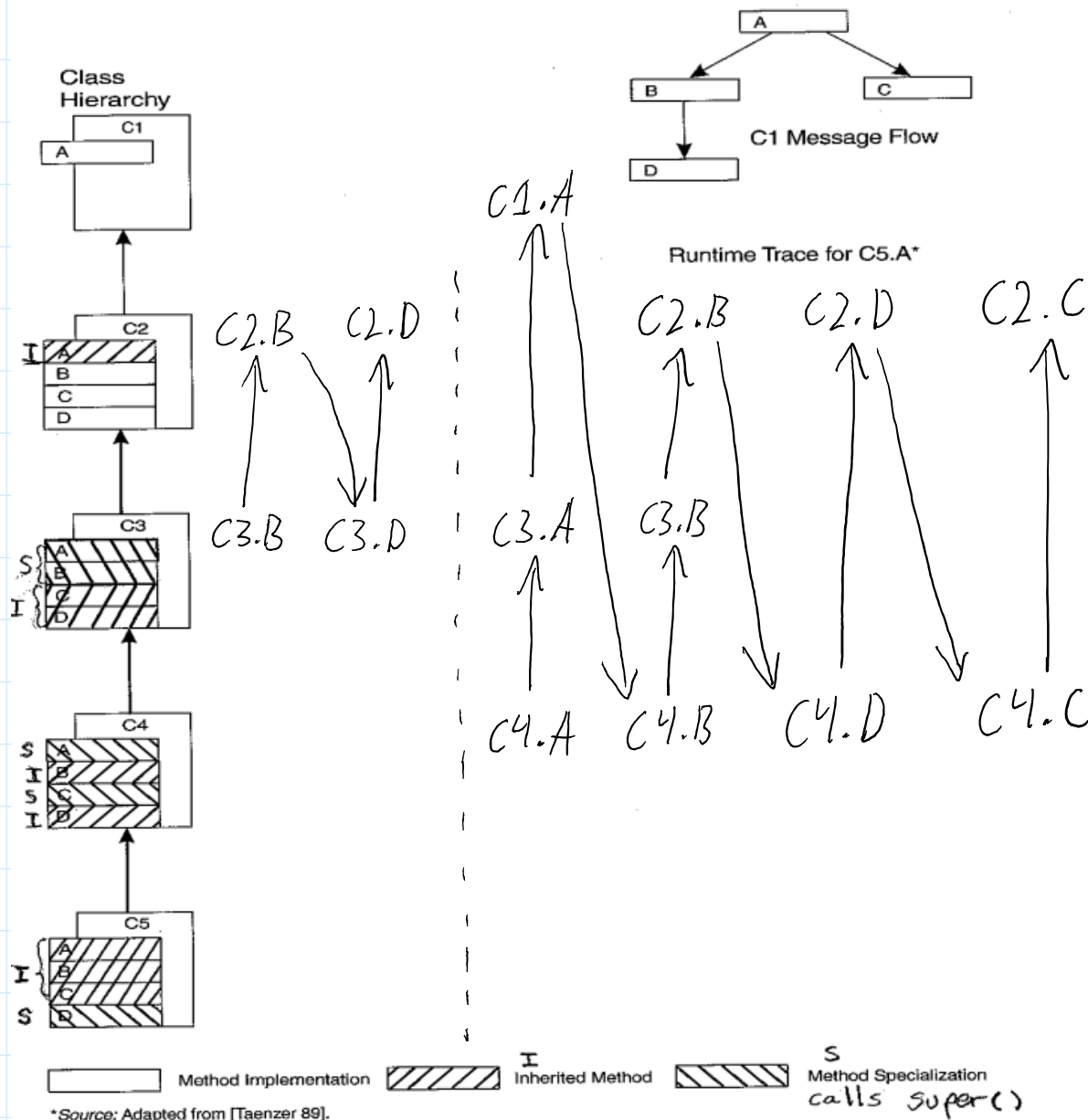
- d) Is 100% NC or 100% EC possible in general? Why, or why not? (2 pts)

In general, it is not possible. Some branches may be infeasible or impossible (e.g. code in a "while(false)" block cannot be exercised).

Sieve(4) Trace (by node):
5, 1, 2, 3, 4, 5, 6, 7,
11, 12, 4, 3, 5, 6, 7, 9, 8,
7, 11, 12, 4, 3, 5, 6, 7, 9,
8, 7, 9, 10, 4, 3, 13, T
Traverses all nodes and edges ⇒ {4} provides 100% NC and 100% EC

Question 2 (10 pts)

- a) Draw the execution of the calls that exhibit the YoYo problem for a runtime trace of C3.B, and for C4.A (Draw both on the diagram below). (8 pts)



- b) Describe what happens when we call C1.D (2 pts)

Depending on the language, calling C1.D will either fail to compile or produce an error, because C1 does not implement or inherit D.

Question 3 (12 pts)


Given the following program:

```
1: public int fibonacci (int i) {  
2:   int fib1 = 1 ;    // fib(n-1)  
3:   int fib2 = 1 ;    // fib(n-2)  
4:   int fib = 0;  
5:   int j;  
  
6:   if (i <= 1)  
7:     fib = 1;  
   else  
8:     for (j=1;  
9:         j<i;  
10:        j++)  
11:       {  
12:         fib = fib2 + fib1 ;  
13:         fib2 = fib1 ;  
14:         fib1 = fib ;  
15:       }  
16:   return fib ;  
17: }
```

Give test cases that will kill the following mutations (4pts each):

(a) Line 6: if (i < 1)


For test case $\bar{T}=1$,

The mutant will produce $\text{fibonacci}(1)=0$ 
The oracle will produce $\text{fibonacci}(1)=1$

\therefore The mutant is killed by test case $\bar{T}=1$

(b) Line 6: if (i == 1)



For test case $\bar{T}=0$,

The mutant will produce $\text{fibonacci}(0)=0$ 
The oracle will produce $\text{fibonacci}(0)=1$

\therefore The mutant is killed by test case $\bar{T}=0$

(c) Line 12: fib2 = fib;

For test case $\bar{T}=3$,

The mutant will produce $\text{fibonacci}(3)=4$ 
The oracle will produce $\text{fibonacci}(3)=3$ 

\therefore The mutant will be killed by test case $\bar{T}=3$