



This exam consists of 8 substantive questions. You have 3 hours – budget your time wisely. Assume the armlab/gcc217 environment unless otherwise stated in a problem.

Do all of your work on these pages. You may use the provided blank spaces for scratch space, however this exam is preprocessed by computer, so for your final answers to be scored you must write them inside the designated spaces and fill in selected circles and boxes completely (● and ■, not ✓ or ✕). Please make text answers dark and neat.

Name: NetID:

Precept:

<input type="radio"/>	P01 - MW 1:20 Xiaoyan Li	<input type="radio"/>	P03 - TTh 12:15 Polly Ren	<input type="radio"/>	P06 - TTh 1:20 Lana Glisic
<input type="radio"/>	P02 - MW 3:30 Nicholas Yap	<input type="radio"/>	P04 - TTh 12:15 David Shustin	<input type="radio"/>	P08 - TTh 3:30 Viola Chen

This is a closed-book, closed-note exam, except you are allowed one two-sided study sheet. Please place items that you will not need out of view in your bag under your working space at this time. Electronic devices, including phones, laptops, smartwatches, smart glasses, etc. may not be used during this exam.

This examination is administered under the Princeton University Honor Code. Students should sit one seat apart from each other and refrain from talking to other students during the exam. All suspected violations of the Honor Code must be reported to honor@princeton.edu.

In the box below, copy **and** sign the Honor Code pledge before turning in your exam:
"I pledge my honor that I have not violated the Honor Code during this examination."

Exam statistics:	10th %ile: 33/65 (50.8%)
Max: 65/65	25th %ile: 41/65 (63.1%)
Mean: 47.9/65 (73.7%)	Median: 49/65 (75.4%)
StdDev: 10.9/65	75th %ile: 57/65 (87.7%)
	90th %ile: 60/65 (92.3%)

X _____

Question 0: First Things First. *That's it.*

0 points

[First College. The "hint" is that it's not hidden at all, its name is right there.](#)

Make sure you have filled out your name, NetID (i.e., armlab login – not PUID, not email alias), precept, and the complete Honor Code pledge text on the front page. Sign your name once you have finished the exam.

The substantive questions begin on Page 3. You should go there now. Ignore all the cryptic problem titles until you are completely finished with the exam; only then should you return to the puzzle at the bottom of this page, if there's time and it interests you.

[The group was the Princeton residential colleges. See explanations within each question.](#)

[Thanks for playing along, and particular kudos to one anonymous student \(unless she wishes to reveal herself\) who quipped: "That's a lot of dad jokes for an exam given on Mother's Day!"](#)

For 0 points (so please don't even think about this if your time would be better spent reviewing your answer to one of the actual questions!), but significant puzzling respect:

- a. This exam has 9 questions, including Q0, with each cryptic title referencing one member of a not-quite-closed class (set) that you're intimately familiar with. What is the group, and which set element goes with each problem? Feel free to share your answers in the blank space above on this page.

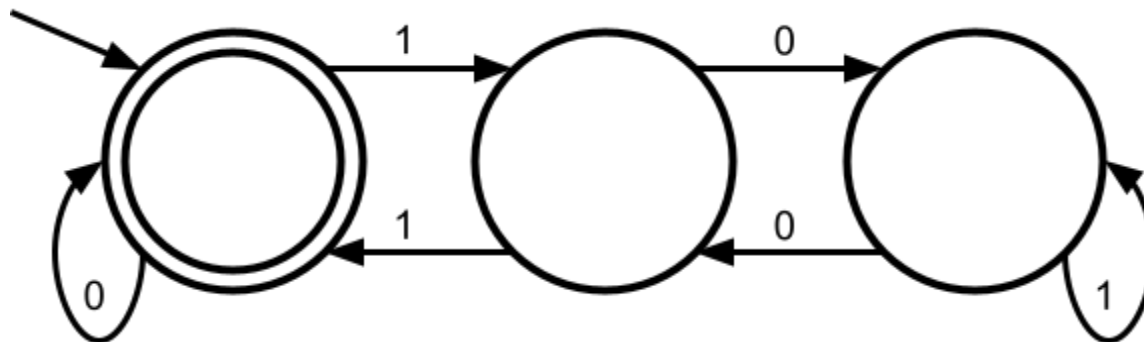
- b. Was that too easy? You're just getting started! Use the other information from the problem titles to **find** each set element (or, in one case, associated term) that has been hidden within its corresponding problem. Feel free, again, to use the space on this page to describe what you've found/solved.

Puzzle hints: The set is ordered; ?s are approximate interpretations; 'A'=65 except when it's 1; you might have to get a problem right to find where the set element is hiding!

Question 1: Princeton Inn the Nation's Service. How *dashing!* 6 points

Princeton Inn was the original name of Forbes College. The hint indicates to look

Consider the following DFA for the first two parts of this question: near the first dash on this page.



- a. What are the first three unsigned integers whose binary representations are accepted by this DFA? Enter the three numbers in the boxes on the right.

0	3	6
---	---	---

- b. In the box below, define the more general set of – or best English description of – binary unsigned integers this DFA accepts. Use no more than 10 words.

Multiples of 3.

(The start state represents $\% 3 == 0$; the middle state represents $\% 3 == 1$; the rightmost state represents $\% 3 == 2$.)

- c. A student brings a LabTA the following line of code to compute the midpoint between lower bound `int a` and upper bound `int b`:

```
mid = (a + b) / 2;
```

The LabTA, remembering a concept from COS 217 and noticing a potential pitfall, suggests that the following computation would be better practice:

```
mid = a + (b - a) / 2;
```

In the box below, explain the problem with the original code that is fixed in the revised version. Use no more than 10 words.

The sum $(a + b)$ could overflow, which the revised version cannot.

The Rockefellers are one of a handful of stand-in names for the class of hyperrich families. In Princeton parlance, we often shorten Rockefeller College to Rocky.

Question 2: Paradigmatic Billionaire, for short, *vertically encoded*. 8 points

Find it vertically, rather than horizontally in Q1. And it's not "encoded", exactly, but in the **code** itself.

a. Consider the following string copy operation using the C standard library:

```
strcpy(pc1, (const char *) pc2);
```

Select all of the following that could be the type of pc1 for this function call to compile cleanly (i.e., without warning or error):

const char * char * const const char * const void *

Recall the signatures for these A3 functions:

```
int SymTable_put(SymTable_T oST, const char *pcKey, const void *pvValue);  
int SymTable_contains(SymTable_T oST, const char *pcKey);  
void *SymTable_get(SymTable_T oST, const char *pcKey);
```

Now consider the following **flawed** testing code for A3's SymTable_put function.

Assume N, oST, copyRet, and FALSE were properly defined and initialized immediately before this snippet and that all necessary header files have been included.

```
char acNameString[N] = "ok";  
SymTable_put(oST, acNameString, "ok");  
copyRet = strcpy(acNameString, "bad");  
if( strcmp("ok", SymTable_get(oST, "ok")) != 0 ||  
    SymTable_contains(oST, copyRet) ) return FALSE;
```

b. In the box below, using no more than 10 words, explain the design error in SymTable_put that this test is intended to catch.

Failing to own/protect its bindings' keys (i.e., not making a "defensive copy")

c. As noted, however, this testing code is incorrect. In the box below, in 1-2 sentences, explain why the testing code will crash in the event that the error it is intended to catch is present.

Without a defensive copy, the strcpy will change the binding's key to "bad", so the subsequent get will not find a binding with the key "ok", and thus will return NULL. strcmp, when passed NULL as an argument, may crash (e.g., fail an assert or dereference the NULL pointer).

Someone quantitatively inclined might be dubbed “Mathy”. The ? indicates we are playing fast and loose with our spelling or pronunciation, so this suggests Mathey College.

Question 3: Quantitatively Inclined? Look at *both sides*. 4 points

The hint indicates that we need to look at both sides of something. In this case, the subsequent paragraph:

Most of the challenge of A4 was to read and understand a large code base, in as much as you could then debug or expand it. Write a COS 217 function comment for this code that details its parameters, behavior, and side effects. Be specific, but answer concisely.

<pre>static int DT_redacted(const char *pcPath, Node_T *poNResult) { Path_T oPPath = NULL; Node_T oNFound = NULL; int iStatus; assert(pcPath != NULL); assert(poNResult != NULL); if(!bIsInitialized) { *poNResult = NULL; return INITIALIZATION_ERROR; } iStatus = Path_new(pcPath, &oPPath); if(iStatus != SUCCESS) { *poNResult = NULL; return iStatus; } iStatus = DT_traversePath(oPPath, &oNFound);</pre>	<pre> if(iStatus != SUCCESS) { Path_free(oPPath); *poNResult = NULL; return iStatus; } if(oNFound == NULL) { Path_free(oPPath); *poNResult = NULL; return NO_SUCH_PATH; } if(Path_comparePath(Node_getPath(oNFound), oPPath) != 0) { Path_free(oPPath); *poNResult = NULL; return NO_SUCH_PATH; } Path_free(oPPath); *poNResult = oNFound; return SUCCESS; }</pre>
--	---

Unlike with function comments for your assignments, for this problem you do not have to discuss the various return values for this function in your comment.

This function searches the DT for a node whose path contents equal the pcPath parameter string. If such a node exists, the out-parameter poNResult is dereferenced to pass back this node. If no such node exists, or the search cannot be completed for various reasons, then *poNResult is set to NULL.

Sir Lucan (King Arthur), Jeeves (P.G. Wodehouse), Alfred (Batman), and Geoffrey (Fresh Prince) are all valets or butlers, suggesting Butler College

Question 4: Lucan, Jeeves, Alfred, or Geoffrey's *Tactile* Mascot. 9 points

Tactile is a common puzzle clue for Braille. Where can we find a series of 3x2 pips?

Consider this Makefile, which executes each of the four stages of the build process separately for a program consisting of three source files: a client (`client.c`) that calls functions from a library interface (`lib.h`) and an implementation of that library (`lib.c`).

```
client: client.o lib.o
    gcc217 client.o lib.o -o client

client.o: client.s
    gcc217 -c client.s

client.s: client.i
    gcc217 -S client.i

client.i: client.c lib.h
    gcc217 -E client.c >| client.i

lib.o: lib.s
    gcc217 -c lib.s

lib.s: lib.i
    gcc217 -S lib.i

lib.i: lib.c lib.h
    gcc217 -E lib.c >| lib.i

clean:
    rm -f client *.o *.s *.i
```

Recall these bash details: `touch` changes the date/time stamp of the given file(s) to the current date/time (as if they had just been edited); `>|` is just like `>` except it overwrites the destination file if it already exists; `rm -f` removes the given file(s); and the wildcard (*) expands to specify all files whose names match the expression it is used in, e.g., `make clean` removes the `client` executable and *all* the `.o`, `.s`, and `.i` files.

Assume that each part on the next page is **independent**, and that for each part the working directory initially contains the three source files, the Makefile, and already contains the seven file targets produced by the Makefile.

For each item below, mark whether the statement is true (T), false (F), or there is not enough information to say for sure (NEI) based on this Makefile and set of files.

The three cells below encode BEE.

- | | T | F | NEI |
|---|----------------------------------|----------------------------------|----------------------------------|
| a. <u>make</u> and <u>make_client</u> are always equivalent
When not given a target, make defaults to the Makefile's first target | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| b. <u>rm -f client.o; make</u> will execute 2 gcc commands
Reassemble client.o, re-link client executable | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| c. <u>client.c</u> has only 1 #include directive: #include "lib.h"
It could also #include standard library headers, which are not listed in dependencies | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> |
| d. <u>touch *.c; make</u> will execute all 7 gcc commands | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| e. <u>make clean; make; make</u> will execute all 7 gcc commands twice
All targets will be up to date from after the second command, so none will be rebuilt | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> |
| f. The assembled object file <u>lib.o</u> will include relocation record(s)
Unknown if there any instructions that required addressing information that the assembler didn't have access to (e.g., an adr of a symbol in a different section, bl to stdlib function) | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> |
| g. <u>touch lib.h; make</u> will execute all 7 gcc commands | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| h. <u>touch lib.c; make</u> will rebuild <u>client.o</u> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> |
| i. Instructions from <u>client.o</u> will appear at lower memory addresses than instructions from <u>lib.o</u> in the final <u>client</u> executable
The relocation stage of the linker may join together the instructions from the various object files' text sections in any order so long as they each remain contiguous. | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> |

You may refer to this abbreviated ARM assembly language reference for Q5-Q8.

Instruction(s)	Description
{add,sub,mul, lsl} dst, src1, src2	dst = src1 {+, -, *, <<} src2
{beq,bne} label	Go to label if comparison was {"equal", "not equal"}
{b,bl} label	{Unconditionally go to , Call function at} label
cmp first, second	Compare first with second, setting bits in PSTATE
ldr dst, [src]	Load 4 or 8 bytes pointed to by src into dst
str src, [dst]	Store 4 or 8 bytes in src to memory pointed to by dst
mov dst, src	Copy contents of register src to register dst
sxtw dst, src (src is w, dst is x)	Copy sign-extended contents of src to dst
ret	Return to address pointed to by x30
R0 - R7 and R0 (w or x)	Used for arguments to and return value from functions

Question 5: Myself of Song of Myself, *think just outside the box*. 7 points

There are a bunch of boxes in this problem. Look just to their right ...

Fill in the box to the left of each description with the letter of the appropriate assembler directive from the options bank on the right.

C Working aliases can be assigned to registers with this directive

E Helps the linker and debugger track the length of a symbol

G int-sized allocation and initialization

D Tells the assembler where subsequent data/code is allocated

F Makes space for a variable in the BSS section

B Allocates and initializes a pointer or long

A Names an immediate value with a semantically meaningful alias

- A. .equ
- B. .quad
- C. .req
- D. .section
- E. .size
- F. .skip
- G. .word

Another ?, another iffy interpretation. The question-word for causal inquiry is “Why”, i.e. “Y”. Anglophone Canadians are stereotyped to use the invariant tag “, eh”. This suggests Yeh College.

Question 6: Causal inquiry north of the border? Be *bold*! 12 points

Find the bolded things in the question, and treat them as indices into the alphabet (A=1, B=2, ... Z=26)

The following assembly language code was generated by gcc217 then very lightly cleaned up for clarity. The original source was a small C function that takes a pointer to a structure as a parameter and returns a long.

```
1      .section .text
2      .global mystery
3      .type mystery, %function
4  mystery:
5      sub    sp, sp, #32
6      cmp    x0, 0
7      bne   .setup
8      mov    x0, 0
9      b     .return
10     .setup:
11     str    x0, [sp, 8]
12     str    x0, [sp, 24]
13     mov    x1, 1
14     str    x1, [sp, 16]
15     b     .check
16
```

```
17     .loop:
18     ldr    w1, [x0, 0]
19     sxtw   x1, w1
20     ldr    x2, [sp, 16]
21     mul   x2, x2, x1
22     str    x2, [sp, 16]
23     ldr    x0, [sp, 24]
24     ldr    x1, [x0, 8]
25     str    x1, [sp, 24]
26     .check:
27     ldr    x0, [sp, 24]
28     cmp    x0, 0
29     bne   .loop
30     ldr    x0, [sp, 16]
31     .return:
32     add   sp, sp, #32
33     ret
```

- a. As noted, the parameter is a pointer to a structure. Based on how the parameter is used in the function, fill out the C struct below, using reasonable field names.

```
struct mysteryArg {
    int val;
    struct mysteryArg *next;
};
```

- b. Translate the assembly language function into idiomatic (not flattened) C code.

```
long mystery(struct mysteryArg *psArg) {
    struct mysteryArg *curr = psArg;
    long product;
    if(psArg == NULL)
        return 0L;
    product = 1L;
    while(curr != NULL) {
        product *= (long) curr->val;
        curr = curr->next;
    }
    return product;
}
```


- b. Consider the **buggy** count function below, which is intended to examine each element of an array of N ARM machine language 3-register add instructions and return the number that use register r for any of the three register operands.

```

size_t count(unsigned int adds[], size_t N, unsigned char r) {
    size_t count = 0;
    size_t index;
    unsigned char mask = 0xf0x1F;
    assert(adds != NULL);
    for(index = 0; index < N; index++) {
        if((adds[index] & mask) == r)
            count++;
        else if(((adds[index] >> 5) & mask) == r)
            count++;
        else if(((adds[index] >> 1716) & mask) == r)
            count++;
    }
    return count;
}

```

Find and fix **four** errors in the code above in-place (add missing code, cross out superfluous code, or cross out and replace incorrect code). If the same error is made multiple times, each occurrence counts separately towards the total of four.

To help you debug, here are some example calls to count on an array `auiArr` with the following instruction machine code values (corresponding assembly language shown beside for ease of understanding); recall `xzr` is encoded as 31.

The three highlighted bytes are the “4th” bytes of the “last 3” instructions. These hex values are ASCII characters N, C, and W.

```

8b1f03e0 (add x0, xzr, xzr)
8b1f001f (add xzr, x0, xzr)
8b00004e (add x14, x2, x0)
8b1f0043 (add x3, x2, xzr)
8b010057 (add x23, x2, x1)

```

(You could also just use the destination registers 14, 3, 23 with the same alphabet code as in Q6 for the same result.)

Sample call	Correct result	This version's result
<code>count(auiArr, 5, 0);</code>	3	4
<code>count(auiArr, 5, 1);</code>	1	0
<code>count(auiArr, 5, 2);</code>	3	3
<code>count(auiArr, 5, 31);</code>	3	0

This one's rough. This name is a reference to the protagonist son in the Farseer Trilogy and other works by Robin Hobb. That's right, he's a Hobb Son ... Hobson College, Question 8: FitzChivalry Farseer? Soon, but construction is *left!* 9 points which will soon join the ranks of the Princeton rescolleges, replacing First (Q0). Look on the left. Hopefully you recall the struct `BigInt` and the `BigInt_T` opaque pointer from A5:

```

struct BigInt {
    /* The number of used digits in the BigInt object */
    long lLength;

    /* The digits comprising the BigInt object.
       aulDigits[0] stores the least significant digit.
       Unused digits are set to 0. */
    unsigned long aulDigits[MAX_DIGITS];
};
typedef struct BigInt *BigInt_T;

```

Of course you also recall the code shown below, from the assembly language function `BigInt_swapDigits`, because you were asked to translate it to C in the *very hard* Q5.

<input type="checkbox"/>	<code>sub sp, sp, 32</code>
<input checked="" type="checkbox"/>	<code>str x0, [sp, A]</code>
<input type="checkbox"/>	<code>str x1, [sp, B]</code>
<input type="checkbox"/>	<code>str x2, [sp, C]</code>
<input checked="" type="checkbox"/>	<code>ldr x0, [sp, A]</code>
<input checked="" type="checkbox"/>	<code>ldr x1, [sp, B]</code>
<input type="checkbox"/>	<code>add x0, x0, 8</code>
<input type="checkbox"/>	<code>ldr x2, [x0, x1, lsl 3]</code>
<input type="checkbox"/>	<code>str x2, [sp, D]</code>
<input checked="" type="checkbox"/>	<code>ldr x0, [sp, A]</code>
<input type="checkbox"/>	<code>ldr x1, [sp, C]</code>
<input checked="" type="checkbox"/>	<code>add x0, x0, 8</code>
<input type="checkbox"/>	<code>ldr x2, [x0, x1, lsl 3]</code>
<input checked="" type="checkbox"/>	<code>ldr x0, [sp, A]</code>
<input type="checkbox"/>	<code>ldr x1, [sp, B]</code>
<input checked="" type="checkbox"/>	<code>add x0, x0, 8</code>
<input type="checkbox"/>	<code>str x2, [x0, x1, lsl 3]</code>
<input checked="" type="checkbox"/>	<code>ldr x0, [sp, A]</code>
<input type="checkbox"/>	<code>ldr x1, [sp, C]</code>
<input type="checkbox"/>	<code>ldr x2, [sp, D]</code>
<input checked="" type="checkbox"/>	<code>add x0, x0, 8</code>
<input type="checkbox"/>	<code>str x2, [x0, x1, lsl 3]</code>
<input type="checkbox"/>	<code>add sp, sp, 32</code>
<input type="checkbox"/>	<code>ret</code>

Some of the instructions in this implementation, however, could be safely removed. Optimize the code above by marking the boxes to remove each instruction for which nothing changes to remaining instructions, nor to the function's behavior, in its absence.