Computer Science 2021 Section 010
Fall 2018
Midterm Exam 1
October 8th, 2018
Time Limit: 50 minutes, 3:35pm-4:25pm

- This exam contains 5 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- Before starting the exam, you can fill out your name and other information of this page, but don't open the exam until you are directed to start. Don't put any of your answers on this page.
- You may use any textbooks, notes, or printouts you wish during the exam, but you may not use any electronic devices: no calculators, smart phones, laptops, etc.
- You may ask clarifying questions of the instructor or TAs, but no communication with other students is allowed during the exam.
- Please read all questions carefully before answering them. Remember that we can only grade what you write on the exam, so it's in your interest to show your work and explain your thinking.
- Students often find that the questions vary in difficulty. Your best strategy is usually to skim over all the questions, and then start working on the ones that look easiest. We also suggest that you leave time at the end to attempt every question, since we can't give you any partial credit if you leave a question blank.
- By signing below you certify that you agree to follow the rules of the exam, not to share exam material with other students before their exams, and that the answers on this exam are your own work only.

The exam will end promptly at 4:25pm. Good luck!

Your name (print): ____________________

Your UMN email/X.500: ____________________`@umn.edu`

Sign and date: ____________________

Question	Points	Score
1	20	
2	31	
3	24	
4	25	
Total:	100	

1. (20 points) C declarations and uses.

   In the left column are ten declarations of a C variable named `x`. In the right column are ten uses of a variable named `x`. However, the way you use a variable depends on its type: most of these uses would be illegal or nonsensical for most of the types. Match each declaration on the left with the legal and most sensible use on the right by writing the letter of the use on the blank. Each choice will be used exactly once.

   The structure `pair` is declared as `struct pair {int a; int b;};`. It is OK if the variable would need to be initialized in between the declaration and the use, even if we haven't shown that. `sinf` is an implementation of the sine function from trigonometry.

(a) ____ `int x[10];`	A. `x.a = 0;`
(b) ____ `int x;`	B. `x = malloc(20 * sizeof(int));`
(c) ____ `struct pair *x[20];`	C. `x[10]->b = x[9]->b;`
(d) ____ `struct pair *x;`	D. `x = '*';`
(e) ____ `char x;`	E. `x->b++;`
(f) ____ `char *x;`	F. `x[12].a = 0;`
(g) ____ `float x;`	G. `x <<= 20;`
(h) ____ `int *x;`	H. `x = sinf(x); /* sine func. */`
(i) ____ `struct pair x[100];`	I. `x = strdup("message");`
(j) ____ `struct pair x;`	J. `int num_elements = sizeof(x)/sizeof(int);`

2. (31 points) Subtraction and overflow.

   Below are the operation tables for unsigned and two's complement subtraction of 3-bit values. An entry in each table shows the result you get if you subtract the value in the column label from the value in the row label. For instance, the number **2** shown in boldface represents that if you perform an unsigned subtraction 3 minus 1, you get 2 (011 - 001 = 010). Some of the entries in the tables are circled, indicating that the results demonstrate unsigned (first table) or signed (second table) overflow. Recall that we say a result has overflowed if it is different from the result that would be computed using unlimited-range mathematical integers.

   We have left some of the table entries blank: your job is to fill them in correctly. You should circle some of the entries you write, according to the same overflow rules. But you shouldn't circle any of the numbers we have written.

   Recommendation: because there are a lot of entries for you to fill in, you probably don't want to do each calculation individually in binary. Instead, do some operations carefully until you see patterns that can let you fill in answers more quickly.

   3-bit unsigned subtraction, with unsigned overflow circled:

$r -^u_3 c$	0	1	2	3	4	5	6	7
0		$\textcircled{7}$	$\textcircled{6}$	$\textcircled{5}$			$\textcircled{2}$	$\textcircled{1}$
1		0		$\textcircled{6}$		$\textcircled{4}$	$\textcircled{3}$	$\textcircled{2}$
2		1		$\textcircled{7}$		$\textcircled{5}$	$\textcircled{4}$	$\textcircled{3}$
3	3	**2**	1					
4	4	3	2					
5	5						$\textcircled{7}$	$\textcircled{6}$
6		5	4	3	2			
7	7	6	5		3		1	0

   3-bit two's complement subtraction, with signed overflow circled:

$r -^t_3 c$	-4	-3	-2	-1	0	1	2	3
-4				-3				$\textcircled{1}$
-3				-2				$\textcircled{2}$
-2								
-1	3			0			-3	-4
0							-2	
1		$\textcircled{-4}$			1			-2
2				3		1	0	
3	$\textcircled{-1}$			$\textcircled{-4}$				

3. (24 points) Matching floating point.

   On the left are descriptions of 8 floating-point numbers. On the right are bit patterns for IEEE single precision floating-point numbers, divided up into groups as the sign bit, the exponent bits, and the fraction bits. Match each description on the left to a bit pattern on the right by filling in the corresponding letter. Each bit pattern is used exactly once.

   This question should not require detailed computations. Instead, try looking for differences between the numbers that lead to differences in their floating-point representations, starting with the easiest (for instance, most distinctive) values first.

(a)	____	10^{-44}	A.	0 11111111 00000000000000000000000
(b)	____	10^{30}	B.	1 00000000 00000000000000000000000
(c)	____	$\pi \approx 3.14159$	C.	0 00000000 00000000000000000000111
(d)	____	$+\infty$	D.	1 10000000 10000000000000000000000
(e)	____	-3	E.	0 10000000 01011011111100001010100
(f)	____	2^{100}	F.	0 11100011 00000000000000000000000
(g)	____	$e \approx 2.71828$	G.	0 10000000 10010010000111111011011
(h)	____	-0	H.	0 11100010 10010011111001011001010

4. (25 points) Pointers.

   Below is some C code that uses pointers in a variety of ways. In the blank next to each call to `printf`, write the number that would be printed. If you want, you may find it helpful to use the space on the right to make notes on what variables contain at each point, or to draw diagrams of what is pointing to what.

```
int a = 10; int b = 20;
int c = 30; int d = 40;
int e = 50;

int *p1 = &a;
p1 = 0;
printf("%d\n", a);           __________

int *p2 = &b;
*p2 += 5;
printf("%d\n", *p2);         __________

int *p3 = &b;
int *p4 = &c;
int *p5 = p4;
p4 = p3;
p3 = p5;
p4 += 10;
printf("%d\n", *p3);         __________

int *p6 = &d;
int **pp7 = &p6;
if (pp7)
    (*p6)++;
else
    *(p6++);
printf("%d\n", **pp7);       __________

int **pp8 = malloc(3 * sizeof(int *));
pp8[0] = &c;
pp8[1] = &d;
pp8[2] = &e;
int ***ppp9 = &pp8;
***ppp9 += ***ppp9;
pp8[1] = pp8[0];
pp8[0] = pp8[2];
pp8[2] = pp8[1];
printf("%d\n", ***ppp9 >> 1);_________
```