QUESTIONS

CS 367 - Computer Systems and Programming

Spring Semester, 2023

Sample Midterm Questions (Wks 0-6): Full Sample Midterm

Questions will be added after each relevant lecture until Midterm.
You should be able to open this at any point in the class and be able to do the questions.
If anything looks scary, this is a great time to review those topics!

The answers will not be provided until just before the Midterm.
Use these sample questions as exercises to help you study!

Changelog:

Welcome to CS367!

- CS 110 and CS 262/CS 222 Review Questions Added (Bases)

Post Week 1 (Jan 23 - Jan 27)

- Chapter 2.1 (CS262/222 Bytes/Boolean/Bitwise Review) Questions Added

Post Week 2 (Jan 30 - Feb 3)

- Chapter 2.2-2.3 Integer-Related Questions

Post Week 3 (Feb 6 - Feb 10)

- Chapter 2.4 Added Questions on Floating Point and Rounding

Post Week 4 (Feb 13 - Feb 17)

- Added Questions on Aggregate Data

Post Week 5 (Feb 20 - Feb 24)

- Fixed a bad answer on Question 12 (0.1111 is 15/16)
- Added an Extra Question on Structs and Questions on Dynamic Memory

Post Week 6 (Feb 27 - Mar 3)

- Added Questions on Processes

1 Data and Machine Representation - Midterm Material

Q1: ..... *0 points*

The full midterm will have questions from the following list of topics, which cover Data Representation (Bytes/Integers/Floating Point), Linking, and Assembly Expressions.

Textbook References The chapter references listed in the topic list below are not exclusive, but are simply the starting point in the textbook where you can review the topic.

Sources Review the slides, recitations, and quizzes for any topics you are not comfortable with.

Topics Topics in this section ***may*** include:

Bitwise and Logical Operations in C

- **(Ch 2.1.1)** Base Conversion, between: Binary, Decimal, and Hexadecimal
- **(Ch 2.1.3)** Addressing and Byte Ordering
- **(Ch 2.1.7)** Bit-Level Operations in C, to include: `|, &, ^, ~`
- **(Ch 2.1.8)** Logical Operations in C, to include: `||, &&, !`
- **(Ch 2.1.9)** Shift Operations in C, to include: <<, >>

Signed and Unsigned Integers

- **(Ch 2.2.1)** Basic Integer Data Types
- **(Ch 2.2.3)** Two's Complement (Signed) Representation
- **(Ch 2.2.3)** Value Ranges on Unsigned Integers: `UMIN, UMAX`
- **(Ch 2.2.3)** Value Ranges on Signed Integers: `TMIN, TMAX`
- **(Ch 2.2.4)** Converting between Signed and Unsigned Representations
- **(Ch 2.2.5)** Signed and Unsigned in C
- **(Ch 2.2.6)** Expanding Signed and Unsigned Bit Representations

Integer Arithmetic

- **(Ch 2.3.1)** Unsigned Integer Addition
- **(Ch 2.3.4)** Unsigned Integer Multiplication
- **(Ch 2.3.2)** Two's Complement Integer Addition
- **(Ch 2.3.5)** Two's Complement Integer Multiplication
- **(Ch 2.3.6)** Multiplication and Division via Shifting

IEEE 754 Floating Point

- **(Ch 2.4.2-2.4.3)** IEEE 754 Representation
- **(Ch 2.4.4)** Rounding

Arrays

- **(Ch 3.8.1)** Array Principles
- **(Ch 3.8.2)** Pointer Arithmetic
- **(Ch 3.8.3)** Nested Arrays (Two-Dimensional Arrays)

Data Structures

- **(Ch 3.9.1)** Structs
- **(Ch 3.9.3)** Alignment
- **(Ch 8.1.0)** Idea of Exceptions
- **(Ch 8.1.1)** Interrupt Vector (a.k.a. Exception Table)
- **(Ch 8.1.2)** Types of Exceptions

Processes

- **(Ch 8.2.1)** Logical Control Flow
- **(Ch 8.2.2)** Concurrent Flows
- **(Ch 8.2.5)** Context Switching
- **(Ch 8.4.2)** Process States, fork, and exit
- **(Ch 8.4.3)** Reaping Child Processes
- **(Ch 8.4.5)** Loading Programs (execv/execl)

2 Sample Questions

These are sample Questions. There are more here than will be on any exam! More will be added each weekend once classes begin.

Q2: ..... *0 points*

Base Conversions: Perform the following Conversions.

(a) Convert the following Unsigned decimal values in to Binary.
There is no limit in bit-width for this problem.

Decimal (Unsigned)	Binary (Unsigned)
12	
25	
15	
68	

(b) Convert the following Unsigned decimal values in to Hexadecimal.
There is no limit in bit-width for this problem.

Decimal (Unsigned)	Hexadecimal (Unsigned)
12	
25	
16	
15	

(c) Convert the following Unsigned Hexadecimal values in to Binary.
There is no limit in bit-width for this problem.

Hexadecimal (Unsigned)	Binary (Unsigned)
15	
10	
2F	
CAB	

(d) Convert the following Unsigned Binary values in to Hexadecimal.
There is no limit in bit-width for this problem.

Binary (Unsigned)	**Hexadecimal (Unsigned)**
1011 1101 0010	
1001 1111 0000	
1011 1101	
11 1101 0010 1001	

Q3: ..... *0 points*

C Programming: Answer the following question about C.
For the given C expressions in Decimal, show the result in 8-bit Binary:

Expression	**Result (8-bit Binary)**
`3 << 1`	
`42 >> 2`	
`15 & 31`	
`13 | 52`	
`24 ^ 42`	
`6 && 1`	
`!0 || !42`	
`~25`	
`-1`	

Q4: ..... *0 points*

Bitwise Operations: Complete a C function that will set bit n to a 1 in a given value and return the updated value. Use bit shifting to create your mask!
As a reference, an **int** is a signed, 32-bit number. So, it has valid bits 0 - 31.

```
int set_bit_x(int value, int n) {



}
```

Q5: ..... *0 points*

Integer Representation: Evaluate the Values of the various Data

(a) Convert the following 8-bit Binary values to Signed and Unsigned Decimals.
This is a combination and extension of two problems (e-f) from the first version of this review

Decimal (Unsigned)	**Binary (Unsigned)**	**Decimal (Signed)**
	1011 0010	
	0000 1111	
	0011 0101	
	1010 1001	
	1111 1111	
	1000 0000	
	1000 0001	

(b) Fill in the Blanks to convert between Signed Decimal and 8-bit Signed Binary.

8-bit Binary (Signed)	**Decimal (Signed)**
1111 1110	
	53
	1

Q6: ..... *0 points*

Operations on Integers: Perform the specified operation.

(a) Fill in the table for the following operations. The Operands are all Unsigned.

Operand (Unsigned Binary)	Expression	Output (Unsigned Binary)
0100	`Extend from 4-bits to 8-bits`	
1010 1101	`Truncate to 4-bits`	
1101	`Extend 4-bits to 8-bits`	

(b) Fill in the table for the following operations. The Operands are all Signed.

Operand (Signed Binary)	Expression	Output (Signed Binary)
0011	`Extend 4-bits to 8-bits`	
0111 0110	`Truncate to 4-bits`	
1011	`Extend 4-bits to 8-bits`	

(c) Fill in the table for the following operations.
Operand and output are all **8-bit** Unsigned Values.

Operand (Unsigned Binary)	Expression	Output (Unsigned Binary)
0001 1010	`>> 4`	
1010 1110	`<< 5`	
1010 1110	`>> 3`	

(d) Fill in the table for the following operations.
Operand and Output are Signed (Two's Complement) **8-bit** Values.

Operand (Signed Binary)	Expression	Output (Signed Binary)
0110 1001	`>> 4`	
1101 1001	`<< 2`	
1101 1111	`>> 6`	

(e) Perform the indicated operation using only Shifts and Arithmetic (<<, >>, +, -).
Use 2 or fewer shifts for each expression

Expression	Expression using only Shifts and Addition
`7*x`	
`18*x`	
`63*x`	
`16*x`	

Q7: ..... *0 points*

Expressions: Consider a 4-bit signed (two's complement) representation.
Fill in the below table for the result of each expression in C. The constants are:

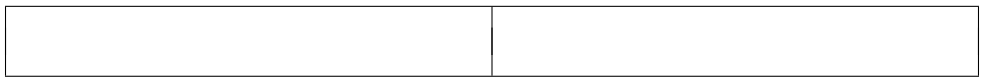
`TMAX` The 4-bit Two's Complement Maximum Value
`TMIN` The 4-bit Two's Complement Minimum Value
`UMAX` The 4-bit Unsigned Maximum Value

Expression in C	Result in Decimal	Result in 4-bit Binary (signed)
(2 << 1) + 1		
TMIN + 1		
~UMAX		

Q8: ..... *0 points*

Perform the following Data Encoding

(a) Store the 16-bit Integer value **0xCAFE** starting at address $0x600$ in Big Endian



(b) Store the 16-bit Integer value **0xFEED** starting at address $0x600$ in Little Endian

Q9: ..... *0 points*

Binary Addition

For each operation on 4-bit Binary Values:

Enter a 1 in the Signed Overflow box if Signed Overflow would occur or a 0 if not.

Enter a 1 in the Unsigned Overflow box if an Unsigned Overflow would occur or a 0 if not.

Operation	Signed Overflow	Unsigned Overflow
`1000 + 1011`		
`0010 + 1011`		
`1001 + 1111`		
`0111 + 0001`		

Q10: ..... *0 points*

C Signed/Unsigned Shifting: Answer the following question about C using these variables.

```
          char a = 0xA4;
          char b = 0x72;
unsigned  char x = 0xA4;
unsigned  char y = 0x72;
```

For the given C expressions in Decimal, show the result in 8-bit Binary:

Expression	Result (8-bit Binary)
`a << 2`	
`b << 2`	
`x << 2`	
`y << 2`	

Expression	Result (8-bit Binary)
`a >> 2`	
`b >> 2`	
`x >> 2`	
`y >> 2`	

Q11: ..... *0 points*

Binary Fractions

Convert the Following Values between Decimal (fractions in 8ths) and Binary

Decimal (in 8ths)	**Binary**
5 3/8	
1/8	
14/32	
14/16	
7/16	
7/8	

Q12: ..... *0 points*

Binary Fractions

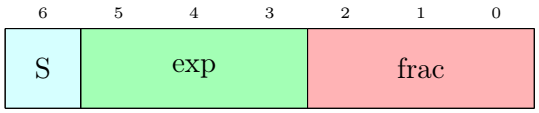
Convert the Following Values between Decimal (fractions in 8ths) and Binary

Decimal (in 8ths)	**Binary**
	1000.111
	0.1111
	0.011
	1.010
	1.10
	1.1

Q13: ..... *0 points*

Floating Point Encodings

For the given 7-bit encoding scheme, answer the following questions:



(a) What is the **bias** in Decimal? (a) _____

(b) What is the smallest Denormalized encoding (in binary)? (b) _____

(c) What is the smallest Positive Denormalized encoding (in binary)? (c) _____

(d) What is the largest Positive Denormalized encoding (in binary)? (d) _____

(e) What is the smallest Positive Normalized encoding (in binary)? (e) _____

(f) What is the largest Positive Normalized encoding (in binary)? (f) _____

(g) What is the encoding for the value 1 (in binary)? (g) _____

(h) What is the encoding for the value Infinity (in binary)? (h) _____

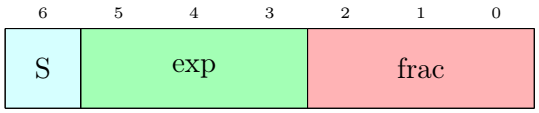
(i) What is the encoding for the value -0 (in binary)? (i) _____

(j) What is an encoding for the value NaN (in binary)? (j) _____

(k) What is the encoding for the value -1 (in binary)? (k) _____

Q14: ..... *0 points*

Floating Point Encoding For the given 7-bit encoding scheme, answer the following questions:



(a) What is the Bias? (a) _____

(b) Fill in the missing fields in this table:

- Except for the Encoding and S columns, all columns will be in Decimal
- `frac` and `M` must be in 8ths. Do not simplify.
- `Value` doesn't need to be simplified beyond $\pm M * 2^E$
- If this is a Special floating point type, only fill in `Value`, `Encoding`, and `S` columns.

Encoding (bit representation)	**S (bit)**	**exp (decimal)**	**E (decimal)**	**frac (8ths)**	**M (8ths)**	**Value** $\pm M * 2^E$
`1 101 110`						
`0 000 001`						
						$-1\frac{3}{8} * 2^3$
	0		**-2**		**7/8**	

Q15: ..... *0 points*

Rounding Round each of the Binary Values to the nearest $\frac{1}{8}$ (3 bits after the binary point) using Round to Nearest Even. Write your answers in Binary.

`101.001101`		`11.0110101`	
`0.001100`		`111.1101`	
`111.11111`		`1.101010`	

Q16: ..... *0 points*

Floating Point Arithmetic

You have two floating point numbers: $A = 5.6$ and $B = 19.2$.

Fill in the tables to perform the intermediate results **in the following formats**:

M_C Represent M_C as a decimal value in the range $[1.0, 2.0)$

E_C Represent E_C as a decimal value representing the exponent for the adjusted M_c.

C Represent C as a decimal value in the form of $(\pm M_c * 2^{E_c})$ where $M_c \in [1.0, 2.0)$

Only M_C, E_C, and C will be graded.

(a) Show the calculation and the resulting value for $C = A * B$.
(Hints: $1.4 * 1.2 = 1.68$, $5.6/2 = 2.8$, $19.2/2 = 9.6$, $9.6/2 = 4.8$)

M_a	
E_a	
M_b	
E_b	
M_c	
E_c	
C	

(b) Show the calculation and the resulting value for $C = A + B$.
(Hints: $1.2 + .35 = 1.55$)

M_a	
E_a	
M_b	
E_b	
M_c	
E_c	
C	

Q17: ..... *0 points*

Floating Point Arithmetic

You have two floating point numbers: $A = 1.6$ **and** $B = 0.75$.

Fill in the tables to perform the intermediate results **in the following formats**:

M_C Represent M_C as a decimal value in the range $[1.0, 2.0)$

E_C Represent E_C as a decimal value representing the exponent for the adjusted M_c.

C Represent C as a decimal value in the form of $(\pm M_c * 2^{E_c})$ where $M_c \in [1.0, 2.0)$

Only M_C, E_C, and C will be graded.

(a) Show the calculation and the resulting value for $C = A * B$.
(Hints: $1.6 * 1.5 = 2.4$)

M_a	
E_a	
M_b	
E_b	
M_c	
E_c	
C	

(b) Show the calculation and the resulting value for $C = A + B$.
(Hints: $1.6 + .75 = 2.35$, $2.35/2 = 1.175$)

M_a	
E_a	
M_b	
E_b	
M_c	
E_c	
C	

Q18: ..... *0 points*

(Arrays) The following arrays are stored **sequentially** starting at address 0x100.

```
int A[20];
char B[40];
int C[200];
char *D[100];
```

Array	Bytes Needed	Starting Address	Equation for &array[i]
A[i]		0x100	
B[i]		0x150	
C[i]		0x178	
D[i]		0x498	

Q19: ..... *0 points*

(Arrays) The following Two-Dimensional arrays are stored **sequentially** starting at address 0x1000.

```
int A[10][10];
char B[40][5];
int C[20][10];
char *D[20][10];
```

Array	Bytes Needed	Starting Address	Equation for &array[i][j]
A[i][j]		0x1000	
B[i][j]		0x1190	
C[i][j]		0x1258	
D[i][j]		0x1578	

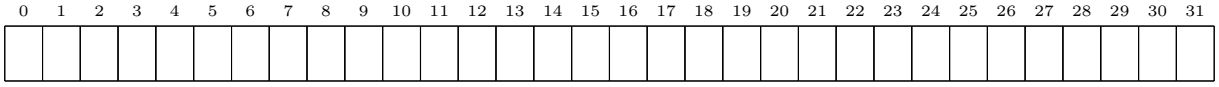
Q20: ..... *0 points*

Structs

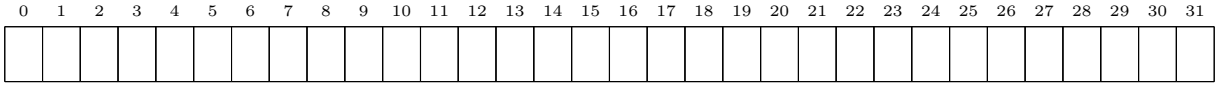
For parts a and b, fill in the charts to indicate which bytes contain each **struct** member. Shade in clearly which byte cells are used for padding. Mark the end of the struct clearly with a vertical line and leave cells after the tail-end, if any, blank.

Once you have completed parts a and b, go on to answer parts c, d, and e.

(a)
```
struct s1 {
  short a;
  long b;
  char c[2];
}
```



(b)
```
struct s2 {
  int a;
  short b;
  struct s1 *c;
  struct s2 *d;
}
```



(c) How many bytes will S1 take up in memory? (c) _____

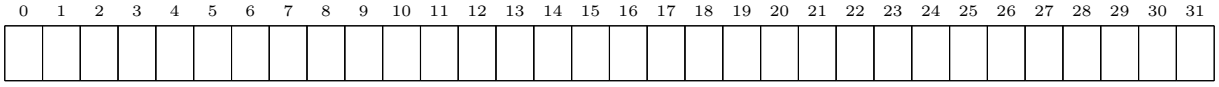
(d) How many bytes will S2 take up in memory? (d) _____

(e) Reorder the Struct s1 into the most optimal ordering. Then draw its new memory allocation.

```
struct s1 {
```

```
struct s1 {
  short a;
  long b;
  char c[2];
}
```

```
}
```



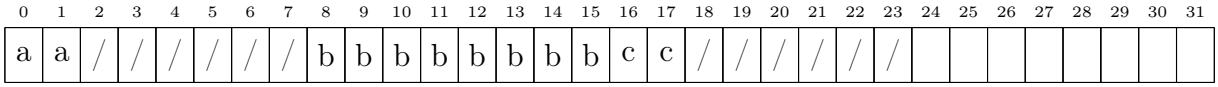
How many bytes will this new S1 take up in memory? (e) _____

Q21: ..... *0 points*

Structs

Given the definition (and layout) for Struct S1, answer the following questions about the Offsets and Sizes for each member of Struct S2

```
struct s1 {
  short a;
  long b;
  char c[2];
}
```



(a)
```
struct s2 {
  short a;
  struct s1 b[3];
  struct s1 *c;
}
```

(b) What offset will S2's **a** member be at in memory? (b) _____

(c) How many bytes will S2's **a** member take up in memory? (c) _____

(d) What offset will S2's **b** member be at in memory? (d) _____

(e) How many bytes will S2's **b** member take up in memory? (e) _____

(f) What offset will S2's **c** member be at in memory? (f) _____

(g) How many bytes will S2's **c** member take up in memory? (g) _____

Q22: ..... *0 points*

Free Block Organization Answer the following questions about Heap Free Block Management (Implicit or Explicit Lists)

(a) Which uses Block Sizes to find the start of the next/previous block? (a) _____ List

(b) Which uses pointers to find the start of the next free block? (b) _____ List

(c) Which is the fastest to find the next free block? (c) _____ List

Q23: ..... *0 points*

Dynamic Memory Allocators Answer the following questions about Implicit List Strategies

(a) Which strategy starts at the beginning and finishes when it finds a fit? (a) _____ Fit

(b) Which strategy starts at the end of the last allocation? (b) _____ Fit

(c) Which results in heavier fragmentation on the front of the list? (c) _____ Fit

(d) Which strategy minimizes fragmentation? (d) _____ Fit

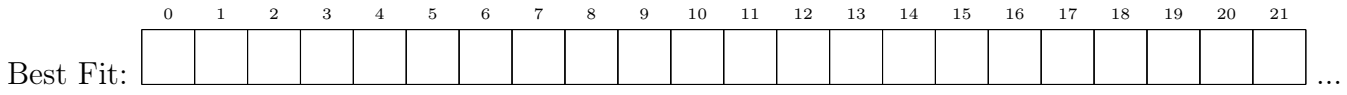
Q24: ..... *0 points*

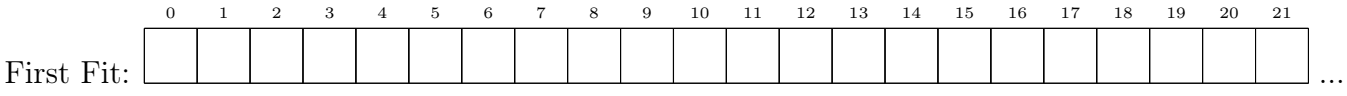
Heap Allocation Handle the following allocate and free requests. Assumptions:

- Each square represents One Word
- Each malloc request is for the number of Words to be allocated.
- **The Heap is Much Larger than Shown**
- Ignore header Words for Size. Assume this is factored into the malloc request already.
- Pad using Double-Word sizes. (ie. Each size is padded to an even number of Words)
- Align using Double-Word alignment. (ie. The starting address must be even)
- If a malloc request is not possible, mark the allocation that it failed on in the code comments.
- For Best-Fit, break ties using the leftmost space.

Sequence:

```
p1 = malloc(4);     Best Fit Fail?  _____

p2 = malloc(4);     Best Fit Fail?  _____

p3 = malloc(2);     Best Fit Fail?  _____

p4 = malloc(6);     Best Fit Fail?  _____

free(p1);

free(p3);

p5 = malloc(2);     Best Fit Fail?  _____

p6 = malloc(4);     Best Fit Fail?  _____
```





Q25: ..... *0 points*

Process Graphs:

(a) Draw the Process Graph for the following code:

```
int main() {
  pid_t pid;
  printf("A");
  if((pid = fork())) {
    waitpid(pid, NULL, 0);
    printf("B");
  }  else {
    printf("C");
  }
  fork();
  printf("D");
}
```

Answer:

(b) Write down two possible **unique** outputs this program may make.

Q26: ..... *0 points*

Processes: Write a C function with the signature `void fun()` to create 4 child processes in a loop and print out the PID of each child as you reap them. For each child process, simply terminate the process once created with an exit code. You do not need to know any of the Macros here.

Make use of the following functions:

- `int fork(void)`
- `pid_t wait(int *status)` // returns the pid of the reaped process

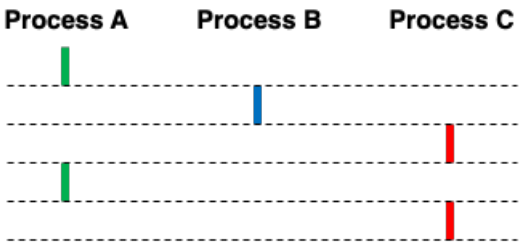
Q27: ..... *0 points*

Processes: For your answer to the previous question, will the processes be printed in order of their creation (eg. 0, 1, 2, 3)?

. 27. _____

Q28: ..... *0 points*

Process Concurrency: Answer the following Questions with either **Concurrent** or **Sequential**



(a) In the above image, what is the relation between A and B? (a) _____

(b) In the above image, what is the relation between A and C? (b) _____

(c) In the above image, what is the relation between B and C? (c) _____

Q29: ..... *0 points*

Processes: Answer the following Questions with a short answer.

(a) What is Reaping?

(a) _____

(b) How does a Parent Process Reap a Terminated child?

(b) _____

(c) What does fork() return for the Parent Process?

(c) _____

(d) What is a Zombie?

(d) _____

(e) What is an Orphan Process?

(e) _____

(f) What Reaps Orphaned Zombies?

(f) _____

(g) What does wait() or waitpid() return?

(g) _____