UC Berkeley CS61C
Spring 2020 Midterm 1

_____ _____

Your Name (first last) *SID*

_____ _____

← SID & Test # To The Left (or aisle) *SID & Test # To The Right (or aisle) →*

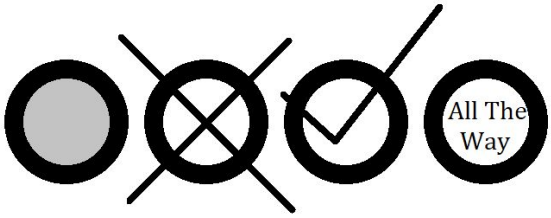
_____

TA name

Make sure to bubble in your answers all the way*…like this:*

● *(select ONE), and* ■*(select ALL that apply)*.

The following are examples of bubbles that *won't* be considered bubbled.



I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that any academic misconduct will be reported to the Center for Student Conduct, and may result in partial or complete loss of credit and penalties that can include an F in the class. I am also aware that Nick Weaver takes cheating personally. I also promise to either give away my first-born child to the kelpies that live in Strawberry Creek or write [sic] after my signature to show that I actually do read the fine print...

Sign your [sic] name: _____

Midterm 1 Clarifications

- 5b) or POSITIVE if the first element is bigger. Also, the numbers themselves are all positive, <2^25, and separated by at least 2.

Q1) *String Cheese* (8 pts = 2 * 1 + 3 + 3)

Mark the correct lines that will allow the program to execute as specified below: There may be multiple correct answers.

a) Correctly gets the number of bytes in a string, including the null-terminator (Mark all that apply)

```
int get_strlen(char* str) {
      ☐ return strlen(str);
      ☐ return strlen(str) + 1;
      ☐ return sizeof(str);
      ☐ return sizeof(str) + 1;
      ☐ return str.strlen() + 1;
      ☐ None of the above
}
```

b) Gets the ith element of an array

```
int get_elem(int* arr) {
      ☐ return arr[i];
      ☐ return arr + i;
      ☐ return *(arr + i);
      ☐ return arr.get(i);
      ☐ return *arr + i;
      ☐ None of the above
}
```

The following code is executed on a 32-bit little-endian system.

```
#include <stdio.h>
int main() {
    int doThis = 0x6C697665;
    char *dont = (char *)(&doThis);
    printf("A: ");
    for (int i = 0; i < 4; i++) {
        printf("%c", dont[i]);
    }
    printf("\n");
}
```

c) What is printed when this program is run? If it crashes/segfaults, write **n/a**.

A:

Carefully read the following code.

```
0  #include <stdio.h>
1  #include <string.h>
2  int main() {
3      char *boo = “go cardinals!”;
4      char *cheer = “go bears!!!!”;
5      printf(“%s”, cheer);
6      for (int i = 0; i < strlen(cheer); i++) {
7          boo[i] = cheer[i];
8      }
9      printf(“%s”, boo);
10 }
```

d) Does the program crash? If the program does not crash, write exactly what is printed to stdout. If the program crashes, identify both the line # that crashes and the line # you would fix to solve the crash

◯ Yes	Line # where the crash occurs ◯ 0 ◯ 4 ◯ 8 ◯ 1 ◯ 5 ◯ 9 ◯ 2 ◯ 6 ◯ 10 ◯ 3 ◯ 7	Line # you would change to solve the crash. ◯ 0 ◯ 4 ◯ 8 ◯ 1 ◯ 5 ◯ 9 ◯ 2 ◯ 6 ◯ 10 ◯ 3 ◯ 7
◯ No	_____	

Q2) *Number RIP* (8 pts = 8 * 1)

Please fill out the following table. Write **N/A** if the conversion is not possible. Some entries have already been filled out for you. You may assume all binary numbers are 8 bits.

Decimal	Binary (Two’s complement)	Octal (base 8, two’s complement)	Hex (two’s complement)	Binary (Biased w/ added bias of -127)
-29				
		131		

Q3) *Unions* (8 pts = 4 * 2)

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

union Fun {
       uint8_t u[4];
       int8_t i[4];
       char s[4];
       int t;
};

int main() {
       union Fun *fun =
         calloc(1, sizeof(union Fun));

       fun->i[0] = -1;

       //Question a
       printf("%u\n", fun->u[0]);

       fun->u[0] *= 2;

       //Question b
       printf("%d\n", fun->i[0]);

       fun->t *= -1;
       fun->t >>= 1;

       //Question c
       printf("%d\n", fun->i[1]);

       fun->s[0] = '\0';

       //Question d
       printf("%d\n", fun->t);

       free(fun);
       return 0;
}
```

Write what each print statement will print out in the corresponding box. Assume that this system is little-endian and that right shifts on signed integers are arithmetic.

a)

_____

b)

_____

c)

_____

d)

_____

Q4) *CS61TREE Memory!* **(8 pts = 7 * 0.5 + 3 + 1.5)**

For this problem, assume all pointers and integers are **four bytes** and all characters are **one byte**. Consider the following C code (all the necessary `#include` directives are omitted). C structs are properly aligned in memory and all calls to malloc succeed. For all of these questions, assume we are analyzing them right before main returns.

```
typedef struct node {
    void *data;
    struct node *left;
    struct node *right;
} node;
```

```
node* newNode(void *data) {
  node *n = (node *) malloc(sizeof(node));
  n->data = data;
  n->left = NULL; n->right = NULL;
  return n; }
```

```
int main() {
  char *r     = "CS 61C Rocks!";
  char s[]    = "CS 61C Sucks!"; /* Reddit review... Warning: Nick sh*tposts too! */
  node nl;
  nl.data = (void *) r;
  node *root  = newNode((void *) &main);
  root->left  = malloc(strlen(r) + 1);
  root->right = newNode((void *) s);
  root->right->left  = newNode((void *) r);
  root->right->right = newNode((void *) &printf);
  root->left  = &nl;
}
```

a) Each of the following evaluate to an address in memory. In other words, they "point" somewhere. Where in memory do they **point**?

	Code	*Static*	*Stack*	*Heap*
root	◯	◯	◯	◯
root->data	◯	◯	◯	◯
root->left	◯	◯	◯	◯
root->left->data	◯	◯	◯	◯
root->right->data	◯	◯	◯	◯
root->right->left->data	◯	◯	◯	◯
&newNode	◯	◯	◯	◯

b) How many bytes of memory are allocated but not **`free()`**d by this program, if any?

_____ Bytes

```
void free_tree(node *n) {
  if (n == NULL) return;
  free_tree(n->left);
  free_tree(n->right);
  free(n);
}
```

c) Given this free function, if we called `free_tree(root)` after all the code in main is executed, this program would have well defined behavior.

◯ True

◯ False

Q5) ***The Bananananananana Hunt*** **(15 pts = 4 + 4 + 7)**

You're on a hunt around campus to find the best fresh banana available. You find a note from the CS61C course staff with clues, but they're encrypted so that only the best students can find the bananas. **Note that the solutions for each part are not dependent on the other parts.**

(a) Your first clue is a string encoded in an integer array `info` of length `len`. We encoded the null-terminated string by placing the *i*th character in the most significant byte of *i*th integer in `info`. Modify the code below so that the original string is properly printed and so that there are **no memory leaks or undefined behavior**.

```
void clue1(unsigned int* info, int len) {
    char* info_to_print = _________________(_____________ sizeof(char));
    for (int i = 0; i < len; i++) {
        info_to_print[i] = (char) ____________________________________;
    }
    printf("%s\n", info_to_print);
    ____________________________________;
}
```

(b) Having discovered the identity, you follow it and find a large array of double precision floating point (type **`double`**). The clue says you want the 5th smallest element casted to an integer. True, you could just go through the array but, being a proper CS student, you decide to first sort the array using a library function and then take the 5th element. Fortunately, C has a quicksort function in the standard library:

```
void qsort ( void * base, size_t num, size_t size,
             int ( * comparator ) ( const void *, const void * ) );
```

That is, the function takes four arguments: a pointer to the array, the total number of elements, the size of each element of the array, and a comparison function. The comparison function should return negative if the first element is less than the second, 0 if they are the same, or 1 if the first element is bigger. Your code should compile without warnings.

```
int comp(void *p1, void *p2){
    double a = ___________________;
    double b = ___________________;
    return ____________________; /* C will cast a double to an int automagically */
}
void clue2(double* info2, int len) {
    qsort(_____________________, ____________________________,
          _____________________, ____________________________);
    printf("%i\n", _____________________________);
}
```

(c) You arrive at the room, only to find a door locked with a keycode. Spray painted on the wall, you see

*"How many stairwells have a power-of-two number of steps? Print the answer **in hex**..."*

So close to your goal, you crowdsource this question to your favorite social media. Enlisting a friend taking CS 186, you end up with an array of step counts for all stairs which are all positive integers. Create a function to see the total number of stairwells with exactly a power of 2. Hint: you know ***X*** is a power of 2 if and only if ***X*** and ***X-1*** have no bits in common and ***X*** is nonzero. You do not need to use all the lines.

```
void pows_of_2(unsigned int* stairs, int len) {
    int matching_entries = 0;
    ____________________________________________________;
    ____________________________________________________;
    for (int i = 0; i < len; i++) {
        ____________________________________________________;
        if (_____________________________________________) {
            matching_entries += 1;
        }
        ____________________________________________________;
    }
    printf(_________________, ___________________ );
}
```

Print format specifier table.

Specifier	Output	Specifier	Output
d or i	Signed decimal integer	**E**	Scientific notation (mantissa/exponent), uppercase
u	Unsigned decimal integer	**g**	Use the shortest representation: %e or %f
o	Unsigned octal	**G**	Use the shortest representation: %E or %F
x	Unsigned hexadecimal integer	**a**	Hexadecimal floating point, lowercase
X	Unsigned hexadecimal integer (uppercase)	**A**	Hexadecimal floating point, uppercase
f	Decimal floating point, lowercase	**c**	Character
F	Decimal floating point, uppercase	**s**	String of characters
e	Scientific notation (mantissa/exponent), lowercase	**p**	Pointer address

Q6) *Fl_at___P_int_u_bers* (9 pts = 3 * 3)

You received a sequence of IEEE standard 16-bit floating point numbers from your friend. So you don't need to look it up on your green sheet, we will remind you that a 16-bit floating point is **1 sign bit, 5 exponent bits, and 10 mantissa bits**. The bias for the exponent is **-15**.

Unfortunately, cosmic rays corrupted some of the data, rendering it unreadable. For the following problems, we will use "x" to refer to a bit that was corrupted (in other words, we don't know what the sender wanted that bit to be). For example, if I received the data "0b0xx1", the sender sent one of "0b0001", "0b0101", "0b0011", or "0b0111".

a) You receive the data "0b0x1x0x1x0x1x0x1x". What is the **hexadecimal encoding** of the **biggest number** the sender could have sent?

0x _ _ _ _

b) You receive the data "0b1110xxxxxxxxxxx". What is the **decimal value** of the **smallest number** the sender could have sent (i.e. it is less than all of the other possibilities)? You must provide the decimal form, ***do not leave as a power of 2***.

_____

c) For the next number, the sign and exponent are correct but all of the mantissa was corrupted. The sender did not send a NaN or infinity. What is the **smallest possible** positive number the sender could have sent **as a power of 2**?

_____

Q7) *RRIISSCC-VV* (12 pts)

In this question, you will implement a simple recursive function in RISC-V. The function takes a decimal number as input, then outputs it's binary representation encoded in the decimal digits.

```
int findBinary(unsigned int decimal) {
      if (decimal == 0) {
            return 0;
      } else {
            return decimal % 2 + 10 * findBinary(decimal / 2);
      }
}
```

For example, if the input to this function is `10`, then the output would be `1010`.

```
findBinary:
    addi sp, sp, -8           # preamble... a0 will have arg and be where we return
    sw ra, 4(sp)
    sw s0, 0(sp)
    beq a0, x0, _____________ # base case, we will just return 0
    ________________________  # set s0 to ??? with a bitwise op
    ________________________  # set a0 to ??? with a bitwise op
    jal ra, _________________ # recursive call
    ________________________  # Load the value 10 into t0
    mul a0, t0, a0            # a0 = a0 * 10
    ________________________  # a0 = ???
postamble:
    ________________________  # Restore ra
    ________________________  # restore s0
    ________________________  # restore sp
end:
    jr ra
```

Good Luck, and ***Don't F@#)(* It Up!***