University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Summer 2016 Instructors: Shreyas Chand, Justin Hsia 2016-07-07

CS61C MIDTERM 1

Last *Name (Please print clearly)*						
First *Name (Please print clearly)*						
Student ID Number						
Circle the name of your Lab TA	**Alex**	**Brian**	**Jinglin**	**Nate**	**Reese**	**Steven**
Name of the person to your: Left | Right						
All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who haven't taken it yet. ***(please sign)***						

Instructions

- This booklet contains 8 pages including this cover page. **The back of each page is blank and can be used for scratch work, but will not be graded** (i.e. not even scanned into Gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats, headphones, and watches. Place *everything* except your writing utensil(s), cheat sheet, and beverage underneath your seat.
- You have 80 minutes to complete this exam. The exam is closed book: no computers, tablets, cell phones, wearable devices, or calculators. You are allowed one page (US Letter, double-sided) of *handwritten* notes.
- There may be partial credit for incomplete answers; write as much of the solution as you can.
- Please write your answers within the boxes and blanks provided within each problem!

Question	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Possible Points	15	12	19	16	12	9	83

If you have the time, **feel free to doodle on this front page!**

SID: _____

Question 1: Number Representation (15 pts)

a) Complete the tables below:

Convert unsigned integers:

Base 8	Hexadecimal
115_8	
	0x1A

Convert to and from IEC prefixes:

Standard	IEC Prefixes
	16 Pebi-bits
2048 students	

b) Due to limitations in storage space, we are using only 4 bits to represent integers.

1) What is the *most negative 2's complement* signed integer (decimal) we can represent?

2) What is the value (decimal) of the 2's complement number `0b1010`?

3) Write a number (binary) that, when added to `0b0100`, will cause *signed overflow*.

c) An amino acid is defined by a set of 3 consecutive nucleotides (A, C, G, or T). For example, ATG is Methionine. All combinations are unique (e.g. ATG ≠ AGT ≠ GTA).

1) How many total *possible* amino acids are there?

2) In reality, there are 21 amino acids found in the human body. How many bits would it take to encode these amino acids in binary?

3) Scientists also use single-digit encodings for amino acids (e.g. 'A' for Alanine). In a single sentence, explain why it is okay that we use A for the amino acid Alanine, the nucleotide adanine, and the hex representation of the decimal number 10.

4) We wish to encode the 21 amino acids in base 2, 3, or 5. Which of these choices allows for the MOST new amino acids discoveries before needing to increase the number of digits and how many new discoveries are allowed in this choice?

Base:	Possible New Discoveries:

SID: _____

Question 2: C Potpourri (12 pts)

a) Given the library function `rand()` that returns a random number between `0` and `(2^32)-1` when called, write a valid C expression that uses *bit operations* `(^, ~, |, &)` to initialize the variable `r` with a random integer between `0` and `n`, which is some power of 2 less than `(2^32)-1`.

```
int n = 8; // In this case, we want r to contain one of {0,1,2,3,4,5,6,7}.

int r = ____________________________________;
```

b) Sally Stanfurd tells you that using a random number generator function is silly in C. She claims that since local variables are not automatically initialized we can use the garbage contained in them as random values.

Does Sally's function produce truly random values (circle one)? **Yes // No**

Briefly explain why or why not.

c) Complete the implementation of the integer array shuffle function below, which randomizes the ordering of the first `n` entries of a given integer array. Each of the first `n` entries in the array should be swapped with an earlier entry exactly once, and the rest of the array should be left unchanged. Assume you have access to a function `random(int r)` that returns a number between `0` and `r-1`, inclusive.

```
void shuffle(int* array, int n) {
      for (int i = n - 1;___________; ___________) {

            _______________________________

            _______________________________

            *(array________) = _______________;

            array[________] = _______________;
      }
}
```

d) Assume integers are 32 bits. Instead of passing in an integer array as expected, we decide to pass in a string to our shuffle function as shown:

```
char str[] = "fee fie foh fum ";
shuffle((int*) str, strlen(str)/sizeof(int));
printf("shuffle result: %s",str);
```

From the following choices, circle ONE of the following if it is a *possible* result of the `printf` statement:

Runtime Error
(e.g. seg fault)

`"fum foh fee fie "`

Compiler Error
(e.g. incompatible pointer types)

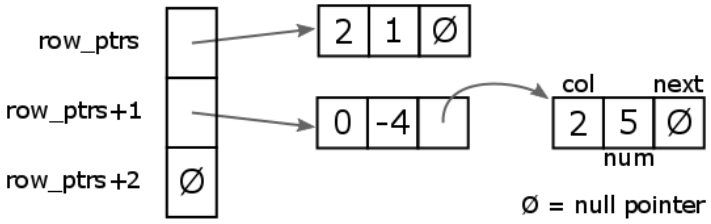
`"ioe h ef uf fmef"`

SID: _____

Question 3: C Structs and Memory (19 pts)

A **sparse matrix** is a matrix in which most of the elements are zero. For matrices of M rows and N columns, instead of storing all $M * N$ entries in an array, we can save memory for large matrices by instead only storing the nonzero entries in linked lists. Here we store an array of pointers to linked lists (`row_ptrs`), one entry for each row of the matrix. Each linked list node will contain (1) the column number, (2) the nonzero integer entry, and (3) a pointer to the next node. Each linked list will be *unsorted*. Both rows and columns are zero-indexed.

For example, the diagram below corresponds to the matrix $\begin{bmatrix} 0 & 0 & 1 & 0 \\ -4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, where $M = 3$ and $N = 4$:



The struct definition for the nodes is given below on the left. On the right are the declarations for both matrix representations: the usual (array) and our new (sparse) designs.

```
struct node {
      int col;
      int num;
      struct node *next;
};
```

Array Matrix Storage

```
int array[];
```

Sparse Matrix Storage

```
struct node **row_ptrs;
```

a) If we have a matrix with $M = 8$, $N = 10$, and 4 nonzero entries, how much memory (in bytes) would each of the following use? Here an integer is 4 bytes and a pointer is 8 bytes.

Array Matrix Storage	Sparse Matrix Storage	
`array`:	`*row_ptrs`:	`struct node`s:

b) Complete the following function `makeNode()`, which creates a new `struct node` for use in other functions and initializes the `col` and `num` fields to the provided arguments. You can ignore running out of memory and you may not need all lines.

```
struct node* makeNode(int column, int number) {

      _____ newNode = _____________________________________;

      ____________________________

      ____________________________

      ____________________________

      return _____________________;

}
```

c) Complete the following function `setNum()`, which sets a nonzero value in our sparse matrix structure. If (row, col) already had a nonzero value, then overwrite the existing value, otherwise add a new `struct node` at the end of the linked list to hold the value.

You should use `makeNode(int column, int integer)` to create any new nodes. You may not need all lines or declared variables.

```
void setNum(struct node **row_ptrs, int row, int col, int num) {
      struct node *prev, *curr;  /* short for previous and current */
      curr = ____________________;
      ___________________________________
      /* insert at front of row */
      if (____________________) {
            ___________________________________
            return;
      }
      /* traverse linked list */
      while (____________________) {
            if (____________________) {
                  ___________________________________
                  ___________________________________
            }
            ___________________________________
            curr = ____________________;
      }
      /* add to end of list */
      ___________________________________
      return;
}
```

SID: _____

Question 4: MIPS Procedures (16 pts)

We wish to write the function `toLower`, which converts a string of letters to lowercase. `toLower` takes in a char pointer, leaves spaces as spaces (assume only letters and spaces), and returns the number of converted characters (including spaces but not including the null terminator). Example: if `strcpy(p,"TeST oNe")`, then `toLower(p)` returns `8` and `*p` now contains `"test one"`.

a) Complete the table below:

	'a'	'z'	'A'	'Z'	Space
Decimal		122	65		32
Binary	0b 0110 0001	0b 0111 1010	0b 0100 0001	0b 0101 1010	0b 0010 0000

b) Based on the information above, fill in the C function prototype below and then write out the C *assignment* that performs the lowercase conversion using *bit manipulation* on a given variable `char c`:

Function prototype: `_____ toLower(_____ p);`

Lowercase conversion: `_____;`

c) Fill in the *recursive* MIPS implementation of `toLower` below, following proper MIPS calling conventions. **Write labels to the left of the provided lines.** You may not need all lines provided. You will be deducted points for using extraneous lines.

```
toLower:
            _____ # prologue

            _____

            _____

            _____ # read char using $a0

            _____ $t0, $zero, lower     # check for null terminator

            _____ # base case

            j   return

            _____ # recursive case; to lowercase

            _____ # store char

            _____ # move to next char

            _____ # recurse

            _____

            _____ # epilogue

            _____

            _____ # return
```

SID: _____

Question 5: MIPS Instruction Formats (12 pts)

For a new microprocessor, we're going to **use a MIPS-like assembly language** but with **16-bit words and instructions**. We want all of the same instruction fields as MIPS, but now need to resize them.

a) Can we still support 32 registers (circle one)? **Yes** // **No**

Provide a *brief* explanation.

b) If we want to keep the J-format instructions (`j` and `jal`) but want a target address field of 12 bits, what is the maximum number of I-format instructions we can support? (Hint: don't forget about R-format)

c) If our immediate field is 9 bits wide, what is the maximum jump *forward* a program could make (relative to the *current* instruction, not the *next* instruction) with a single branch instruction? **Answer in number of instructions.**

d) Using the following arbitrary choices for field sizes, translate the following MIPS-like instruction into machine code for our new microprocessor. Assume the same instruction and register numbering systems as MIPS.

```
srl $2, $3, 2
```

opcode (3)	rs (2)	rt (2)	rd (2)	shamt (4)	funct (3)

e) Stanley Stanfurd claims that for 16-bit words, our shamt field *must* be 4-bits wide, but we counter that we can get away with fewer (e.g. 2 bits). Briefly explain why.

SID: _____

Question 6: Running a Program (9 pts)

The stored-program concept revolutionized computing by allowing machines to be multipurpose (i.e. run many different programs) instead of specialized or hard-coded.

a) Circle all that apply. If I decide to write my program in a compiled language instead of an interpreted language, then I would expect the program to be:

FASTER SLOWER LARGER SMALLER PORTABLE NON-PORTABLE

b) Which stage in CALL actually creates the Code section in memory?

c) Which stage in CALL allows the reuse of other people's code?

d) What information in an object file allows for the displacement of instructions in the Code section of memory when multiple files are combined in the executable?