This practice exam is based on the actual Midterm #1 exam from Autumn 2017.

CS107  
Spring 2017  
Cynthia Lee

CS107 Midterm #1 Examination (Practice #1)

This is a closed book, closed note, closed computer exam. You have **90 minutes** to complete all problems. You don’t need to `#include` any libraries, and you needn’t use `assert` to guard against any errors. Understand that the majority of points are awarded for concepts taught in CS107, and not prior classes. You don’t get many points for `for`-loop syntax, but you certainly get points for proper use of `&`, `*`, and the low-level C functions introduced in the course. **The last page of the exam is a reference sheet.** DO **NOT** ADD OR REMOVE PAGES OF THE EXAM. If you need extra space for scratch and/or problem responses, use the blank back sides of each page. Changing the pages confuses the online grading system and will cause you to lose points.

SUNet ID (myth login): ______________________________________  @stanford.edu

Name: ______________________________________________________

I accept the letter and spirit of the honor code. I will neither give nor receive unauthorized aid on this exam.

[signed] ____________________________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Integer representation</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Memory diagram</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Strings and pointers</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. CVector</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>44</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1: Integer Representation (10pts)
There is a small amount of scratch space between problems for you to write your work, but it is not necessary to show your work, nor will it be evaluated. (You may also use the blank back sides of each exam page as scratch space.) Please write your answer on the provided lines.

a) Write the signed (two’s complement) binary number 11101100 in decimal: _____________

b) Write the unsigned binary number 1100011011 in hexadecimal: _____________________

c) Write the hexadecimal number 0x1DEAD as an unsigned binary number:
______________________________________________________

______

____

d) Write the decimal number 21 as an 8-bit binary number: ____________________________

e) Write the decimal number -15 as an 8-bit signed (two’s complement) binary number:
______________________________________________________

__________

_________________
Problem 2: Memory Diagram (10 points)

For this problem, you will draw a memory diagram of the state of memory (like those shown in lecture) as it would exist at the end of the execution of this code:

```c
int eleven = 11;
char *stranger = "things";
int **upside = malloc(3 * sizeof(int*));
upside[0] = malloc(4);
*upside[0] = 2;
upside[1] = &eleven;
upside[2] = (int*) ((char*)stranger + 1);
```

Instructions:
- Place each item in the appropriate segment of memory (stack, heap, read-only data).
- Please write array index labels (0, 1, 2, ..) next to each box of an array, in addition to any applicable variable name label. (With the array index labels, it doesn’t matter if you draw your array with increasing index going up or down—or sideways for that matter.)
- Draw strings as arrays (series of boxes), with individual box values filled in appropriately and array index labels as described above.
- Take care to have pointers clearly pointing to the correct part of an array.
- Leave boxes of uninitialized memory blank.
- NULL pointer is drawn as a slash through the box, and null character is drawn as '\0'.

```
Stack
```
```
Heap
```
```
Read-only Data
```
Problem 3: Strings and pointers (10 points)

(a) (4pts) Consider the following code, compiled using the compiler and settings we have been using for this class.

```c
char *str = "Stanford University";
char a = str[1];
char b = *((char*)((int*)str + 3));
char c = str[sizeof(void*)];
```

What are the char values of variables a, b, and c? (a is filled in for you as an example) Write “ERROR” across the box if the line of code declaring the variable won’t compile.

- a: 't'
- b: 
- c: 

(b) (6pts) The code below has three buggy lines of code in it. The three buggy parts of the code are noted in bold. Next to each buggy line, write a new line of code that fixes the bug. You may have an idea for restructuring the program that would also fix the bugs, but you must only write code to replace the lines shown in bold—one line of replacement code per one line of buggy code.

The purpose of this function is to take an array of strings (always size 3) and returns a heap-allocated array of size 2, where the first entry is the concatenation of the first two strings in the input array, and the second entry is a copy of the third string in the input array. The two strings in the returned array are both newly allocated on the heap. The input is not modified in any way. You may assume that the input is always valid: the array size is always 3, none of the array entries is NULL, and all strings are valid strings.

```c
char **pair_strings(char **three_strings) {
    char *return_array[2];

    size_t str0len = strlen(three_strings[0]);
    return_array[0] = malloc(str0len + strlen(three_strings[1]));

    strcpy(return_array[0], three_strings[0]);
    for (size_t i = 0; i < strlen(three_strings[1]); i++) {
        for (________________________) {
            return_array[0][str0len + i] = three_strings[1][i];
        }
    }
    return_array[1] = strdup(three_strings[2]);
    return return_array;
}
```
Problem 4: CVector (14 points)

(a) (7pts) For this problem, you will write a function that takes an array as input and returns a CVector. Specifically, this function takes an array of int* (that is an array of pointers-to-ints), and the number of elements in the array. Each pointer in the input array points to exactly one heap-allocated integer (assume the size provided as input is accurate, and each pointer points to a different heap-allocated int).

Your function should: (1) create a CVector that holds type int (just plain int), (2) add the integer values pointed-to by the pointers in the input array to the CVector, then (3) free each integer in the input array, and free the array itself. The CVector functions are listed on the last page of this exam, for your reference. The order of the CVector should match the order of the input array. The challenge for this question is to be really careful with your levels of indirection!

CVector *ptrarr_to_intvec(int *arr[], size_t nelems) {
(b) (7pts) Now write a function that takes the return value of `ptrarr_to_intvec()`, sorts it using the usual CVector sort facility, and prints the contents in sorted order (ascending order, one integer per line). You will need to write a callback function to pass to the sort. Space is provided for you below (`my_compare`). You may want to consult the Reference Page at the end of the exam.

```c
// Callback
int my_compare(const void *addr1, const void *addr2)
{
}

// Sort and print
void sort_and_print(int *arr[], size_t nelems)
{
    CVector *cv = ptrarr_to_intvec(arr, nelems);
    //sort cv:

    // print the contents of cv (iterate over contents and print):
}
```
CVector Functions

typedef int (*CompareFn)(const void *addr1, const void *addr2);
typedef void (*CleanupElemFn)(void *addr);
CVector *cvec_create(int elemsz, int capacity_hint, CleanupElemFn fn);
void  cvec_dispose(CVector *cv);
int   cvec_count(const CVector *cv);
void *cvec_nth(const CVector *cv, int index);
void  cvec_insert(CVector *cv, const void *addr, int index);
void  cvec_append(CVector *cv, const void *addr);
void  cvec_replace(CVector *cv, const void *addr, int index);
void  cvec_remove(CVector *cv, int index);
int   cvec_search(const CVector *cv, const void *key,
                     CompareFn cmp, int start, bool sorted);
void  cvec_sort(CVector *cv, CompareFn cmp);
void  *cvec_first(CVector *cv);
void  *cvec_next(CVector *cv, void *prev);

CMap Functions

typedef void (*CleanupValueFn)(void *addr);
CMap *cmap_create(size_t valuesz, size_t capacity_hint, CleanupValueFn fn);
void cmap_dispose(CMap *cm);
int cmap_count(const CMap *cm);
void cmap_put(CMap *cm, const char *key, const void *addr);
void cmap_get(const CMap *cm, const char *key);
void cmap_remove(CMap *cm, const char *key);
const char *cmap_first(const CMap *cm);
const char *cmap_next(const CMap *cm, const char *prevkey);

Other Functions

void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
void *memset(void *ptr, int value, size_t num);
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
char *strdup(const char *s);
char *strcat(char *dest, const char *src);
char *strchr(char *str, int character);
char *strtok(char *str, const char *delimiters);
int sprintf (char * str, const char * format, ...);
int strcmp(const char *str1, const char *str2);
int strncmp(const char *str1, const char *str2, size_t n);