Fall Semester 2016

KAIST EE209

Programming Structures for Electrical Engineering

Mid-term Exam

Name: ______________________________

Student ID: ______________________________

This exam is open book and notes. You should not _share_ any books/notes with your students during the test. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. You have 165 minutes (9:00 AM – 11:45 AM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 10
Question 2	/ 20
Question 3	/ 20
Question 4	/ 25
Question 5	/ 25
Question 6	/15 (extra credit)
Total	/ 115

Name: Student ID:

1. (10 points) Small Programs
   - Assume that we have included proper header files (e.g., <stdio.h>).
   - Assume that we are using 64-bit OS.

   (a) (2 points) What's the output of this code snippet?

```
char *p, *q = p;
char x = 'A';

p = &x;
printf("*q = %c\n", *q);
```

   (b) (2 points) What's wrong with the code below? Briefly explain the reason why it is wrong.

```
char c;

while ((c = getchar()) != EOF)
    putchar(c);
```

Name: Student ID:

(c) (3 points) What’s the output of this code snippet?
(%zu takes an unsigned long integer)

```
 float f[] = {2016.3, 98.4};
 float *pf = f;

 printf("1:%zu 2:%zu 3:%zu\n", sizeof(f), sizeof(pf),
sizeof(*pf));
```

(d) (3 points) What’s the output of this code snippet?

```
int f(int x)
{
  assert(x >= 0);
  if (x == 0 || x == 1) return x;
  return f(x-1) + f(x-2);

}

…
printf("%d\n", f(5));
```

Name: Student ID:

2. (20 points) Mergesort

We implement mergesort in this problem. Mergesort is a fast sorting algorithm similar to quicksort. It works by dividing a list of elements into two halves, recursively sorting each half, and merging the sorted two halves into one sorted list. At a given stage with a list of n elements, it divides the list into two lists, say A and B, such that the number of elements in each list is equal or differs by only one element, and recursively sorts A and B, and merges A and B into one sorted list. Here is the skeleton code of mergesort that you need to complete.

```
void MergeSortedList(int a[], int start, int mid, int end)
{
    // (1) You need to fill out this function
}


void MergeSort(int x[], int start, int end)
{
   int mid;


   // (2) Handle the base condition here


   mid = (start + end) / 2;
   MergeSort(x, start, mid);     // sort x[start] to x[mid]
   MergeSort(x, mid + 1, end);   // sort x[mid+1] to x[end]
   // merge two sorted lists into one sorted list
   MergeSortedList(x, start, mid, end);
}
```

(a) (5 points) Write the code for (2). (2) checks if it’s the base case, and handles the base case if so. (Hint: it requires only one or two lines depending on your coding style.)

Name: Student ID:

(b) (15 points) Write the code for (1). (1) merges two sorted lists (a[start] … a[mid], and a[mid+1] … a[end]) into one sorted list. It works as follows.

Let's say A and B are the two sorted lists (A: a[start..mid], B: a[mid+1..end])

Say C is a temporary array that can hold all elements in A and B (e.g., C can hold 'end – start + 1' elements)

Do the following until C is full

- Retrieve the current smallest numbers each from list A and B.
- Store the smaller number of the two to C
- Remove the stored number from the list where it was drawn from.

Copy C back to A and B.

```
void MergeSortedList(int a[], int start, int mid, int end)
{
  int n = end - start + 1;       // # of elements in A and B
  int c[n];                      // allocate an array for C
  int i = 0;                     // c's current index
  int i1 = start, i2 = mid + 1; // starting index for A and B

  while (i < n) {
     int x = a[i1];  // retrieve the current smallest number in A
     int y = a[i2];  // retrieve the current smallest number in B
     if (x < y) {
       c[i++] = x;
      if (i1 == mid) { // A is depleted?
```

Blank X

```
      }
      i1++;  // move on to the next smallest element in A
     } else {
            c[i++] = y;
```

Blank Y

```
         }
       } // end of while
       // copy C back to A and B
```

Blank Z

```
    }
```

Please fill out the blanks X, Y, and Z above.

Name: Student ID:

3. (20 points) Programming a word dictionary

Here is the source code listing.

- strdup(key); copies an inputstring (key) into a newly-allocated memory and return its pointer. It returns NULL if memory allocation fails.

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

#define TRUE        1
#define FALSE       0
#define DICTSIZE 1024

struct Node {
  char *key; char *canonical; struct Node* next;
};
struct Dict {
  struct Node *b[DICTSIZE];
};

int HashString(const char *k, int size)
{
  unsigned int t = 0U;
  while (*k != 0) t = t * 65599 + *k++;
  return (t % size);
}
int CompareChar(const void *a, const void *b)
{
  char p = *(char *)a; char q = *(char *)b;
  return (p - q);
}
```

Name: Student ID:

```
struct Dict *DictCreate(void)
{
  return calloc(1, sizeof(struct Dict));
}
int DictInsert(struct Dict *t, char *key)
{
  struct Node *p = malloc(sizeof(struct Node));
  int h;

  if (p == NULL) return FALSE;
  if ((p->key = strdup(key)) == NULL) return FALSE;
  if ((p->canonical = strdup(key)) == NULL) {
      free(p->key); return FALSE;
  }
  p->next = NULL;

  qsort(p->canonical, strlen(key), 1, CompareChar);
  h = HashString(p->canonical, DICTSIZE);
  p->next = t->b[h];
  t->b[h] = p;
  return TRUE;
}

int DictPrint(struct Dict *t, char *key)
{
  char *canonical;
  struct Node *p;
  int found = FALSE;
  int h;

  canonical = strdup(key);
  if (canonical == NULL) return FALSE;
```

Name: Student ID:

```
    qsort(canonical, strlen(key), 1, CompareChar);
    h = HashString(canonical, DICTSIZE);
    p = t->b[h];

    for (p = t->b[h]; p != NULL; p = p->next) {
        if (strcmp(p->canonical, canonical) == 0) {
            printf("%s\n", p->key);
            found = TRUE;
        }
    }

    free(canonical);
    return found;
}

int main()
{
    struct Dict *t = DictCreate();
    if (t == NULL) {
        fprintf(stderr, "DictCreate() failed\n");
        return -1;
    }
    DictInsert(t, "pots");
    DictInsert(t, "hello");
    DictInsert(t, "post");
    DictInsert(t, "stop");
    DictInsert(t, "tops");

    DictPrint(t, "psot");
    return 0;
}
```

Name: Student ID:

(a) (5 points) What's the output of the program? Assume all C runtime library function calls succeed.

(b) (5 points) Explain what DictPrint(struct Dict *t, const char *key) does in plain English. What's the algorithm used here?

(c) (10 points) Write DictFree(struct Dict *t); Note that it should not have any memory leak.

```
void DictFree(struct Dict* t)
{










}
```

Name: Student ID:

4. (25 points) Key-value storage with a hash table

We learned a hash table-based key-value storage in class. Please assume the same data structures (struct Table, struct Node) and functions (Table_create(), Table_search()) as in class. For your reference, Table_add() is shown as below (identical in the lecture slides)

```
int Table_add(struct Table *t, const char *key, int value)
{
   struct Node *p =(struct Node*)malloc(sizeof(struct Node));
   int h = hash(key);
   if (p == NULL) return FALSE;
   p->key = key;
   p->value = value;
   p->next = t->array[h];
   t->array[h] = p;
   return TRUE;
}
```

In a separate C file, we write the following code:

```
char p1[100] = "EE209";
char p2[100] = "EE205";
struct Table *t;
int found, value;

t = Table_create();    // assume that Table_create() succeeds
Table_add(t, p1, 3);   // assume that Table_add() succeeds
Table_add(t, p2, 10);  // assume that Table_add() succeeds
strcpy(p1, p2);
found = Table_search(t, "EE209", &value);  // (1)
found = Table_search(t, "EE205", &value);  // (2)
```

Name: Student ID:

(a) (5 points) Will Table_search() in (1) succeed? If it succeeds, what do you see in the variable, “value”? Briefly justify your answer.

(b) (5 points) Will Table_search() in (2) succeed? If it succeeds, what do you see in the variable, “value”? Briefly justify your answer.

(c) (5 points) Describe the problem in the code above.

Name: Student ID:

(d) (10 points) Rewrite Table_add() to fix the problem. For safety, what code do you need to add to Table_free()? Briefly explain it in English.

```
int Table_add(struct Table *t, const char *key, int value)
{








}
```

Name: Student ID:

5. (25 points) Printing all combinations of a string

We are writing a function that prints out every possible combinations of the characters in a string (for a string with length n, there is n! possible combinations). For example, if the input string is “abc”, your function should print out

```
abc
acb
bac
bca
cab
cba
```

The number of output lines should be n! for a string with length n. This means that you do not need to make each output line unique. For example, for “aaa”, you will see six (=3!) lines of “aaa” in the output.

(a) (10 points) Please describe the algorithm in plain English. Please be specific in each step. (Hint: use a recursive function)

(b) (15 points) Fill out the function below. You may define and use other functions if needed. Also, you can use any C runtime library functions (no #include is needed).

```
void PrintAllCombination(const char *s)
{
    char *p = strdup(s);
    if (p == NULL) return;





    free(p); // free the newly allocated string memory
}
```

Name: Student ID:

6. (15 points) Extra credit

   (a) (5 points) What is the time complexity of mergesort in terms of O notation (problem 2) if the number of elements to sort is n? Briefly explain the reason for your answer.

   (b) (10 points) You have (32 * N) light bulbs to monitor. Each light bulb is given an identifier from 0 to 32 * N -1. Value of 0 represents that the light bulb is off while 1 represents that the light is on. To save memory, you keep the status of light bulbs in a bitmap, and write two functions, IsLightOn() and SetLight(). IsLightOn(int id) returns the current value for light bulb identifier, id. SetLight(int id, int value) sets the light bulb of identifier, id, to value. Fill out these functions (Hint: Use bitwise operators)

```
unsigned int bulb[N]; // 32 * N bit locations

int IsLightOn(int id)

{






}

void SetLight(int id, int value)

{






}
```