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);
```

⇨ The output could be a random value. The program could crash since dereferencing q may cause access violation.

   (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);
```

⇨ c should be declared as int since it needs to represent EOF in addition to all the characters in the ASCII code table.

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));
```

⇨ 1:8 2:8 3:4

(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));
```

⇨ 5 (f(0) = 0, f(1) = 1, f(2) = 1, f(3) = 2, f(4) = 3, f(5) = 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 merge 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 the elements from x[start] to x[mid]
    MergeSort(x, mid + 1, end);        // sort the elements from x[mid+1] to x[end]
    MergeSortedList(x, start, mid, end); // merge two sorted lists into one sorted list
}
```

(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.)

⇨ if (start >= end) return;

(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 is 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’s size is end – start + 1)

Do the following until C is full

- Retrieve the current smallest number 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;
    int c[n];
    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) { // is A depleted? (blank X)
                memcpy(c + i, a + i2, (end – i2 + 1) * sizeof(int));
                // or
                // while (i2 <= end) c[i++] = a[i2++];
                break; // or i = n;
            }
            i1++;
        } else {
```

```
c[i++] = y;
// (blank Y)
if (i2 == end) {
    memcpy(c + i, a + i1, (mid – i1 + 1) * sizeof(int));
    // or
    // while (i1 <= mid) c[i++] = a[i1++];
    break; // or i = n;
}
i2++;
}

} // end of while
// copy C back to A and B (blank Z)
memcpy(a + start, c, n * sizeof(int));
// or
// for (i = 0; i < n; i++) a[start + i] = c[i];
}
```

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.

```
#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);
}
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;

    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;
}
```

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

⇨

tops

stop

post

pots

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

⇨ It prints out all anagrams of the string, key in t. String A and B are called anagrams if the two sets of the characters used in A and B are identical. The algorithm here is to keep a canonical string of a given string, which is an anagram to the string so that the characters in the canonical string are sorted in their alphabetical order (in the order of the ASCII code table). If the canonical string of an item is the same as the input key, its original string (p->key) is printed out.

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

```
void DictFree(struct Dict* t)
{
  struct Node*p, *next;
  int i;

  for (i = 0; i < DICTSIZE; i++) {
      for (p = t->b[i]; p != NULL; p = next) {
        next = p->next;
         free(p->key); free(p->canonical); free(p);
      }
  }
  free(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 t is assigned a non-null pointer
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)
```

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

⇨ No. The table has a key that points to p1, but its content is changed to “EE205”. So, Table_search() won’t find an entry with the key, “EE209”.

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

⇨ Yes, and the value will be 10. The table keeps two entries whose key is “EE205”. However, the right value will be returned since the correct entry is stored later than (p1, 3). That is, even if “EE209” and “EE205” are hashed into the same bin, since (p2, 10) is inserted later than (p1, 3), it will be accessed earlier than (p1, 3). So 10 will be returned as value.

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

⇨ The source of the problem is the fact that the table keeps a pointer to the key string. If the key string is updated outside the table, the key in a table entry is changed as well, pointing to a wrong key string.

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

```
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 = strdup(key);
    if (p->key == NULL) return FALSE;
    p->value = value;
    p->next = t->array[h];
    t->array[h] = p;
    return TRUE;
}
```

⇨ The only two lines that need to be changed are boldfaced above. In Table_free() we need to make sure that p->key is freed before p.

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)

- ⇨ There are many variants, but here is one solution. For an n-character string, p, do the following:

if n is 1, print the string and finish.

if n is larger than 1,

Call PrintCombination with p, len = n, k = 0 as arguments.

PrintCombination prints our all combinations while fixing the first k characters of p. Given (p, len, k), PrintCombination iterates through all characters that can be the (k+1)-th character, and for each case, it calls itself with (p, len, k+1) to print out all combinations while fixing the first k+1 characters. When k is len -2, it means the function can change only the last two characters in the string. So, it prints out the two possible combinations and return (no more recursive call).

Name: Student ID:

(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 Swap(char *a, char *b)
{
    char t = *a;
    *a = *b;
    *b = t;
}


void PrintCombination(char *s, int n, int k)
{
    int i;
    assert(k <= (n-2));

    if (k == (n - 2)) {
        printf("%s\n", s);
        Swap(s + k, s + k + 1);
        printf("%s\n", s);
        Swap(s + k, s + k + 1);
        return;
    }
    PrintCombination(s, n, k+1);
    for (i = k + 1; i < n; i++) {
        Swap(s + k, s + i);
        PrintCombination(s, n, k + 1);
        Swap(s + k, s + i);
    }
}
```

```
void PrintAllCombination(const char *s)
{
        char *p = strdup(s); // copy the string into a new memory location
        if (p == NULL) return;

        n = strlen(p);
        if (n == 1) {
           printf("%s\n", s);
            free(p);
           return;
        }

        PrintCombination(p, n, 0);
        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.

⇨ O(n log n). There are (log n) stages of list division, and each stage traverses the entire list once (n times). So it's O (n log n) steps required to complete the entire algorithm.

(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

```
unsigned int bulb[N]; // we have 32 * N bit locations in the array bulb

int IsLightOn(int id)
{
   int n = id / 32;
   int off = id % 32;
   return ((bulb[n] >> off) & 0x1);
}


void SetLight(int id, int value)
{
   int n = id / 32;
   int off = id % 32;
   if (value == 1)
        bulb[n] |= (0x1 << off);
   else
        bulb[n] &= ~(0x1 << off);
}
```