Spring Semester 2022

KAIST EE209

Programming Structures for Electrical Engineering

Mid-term Exam

Name: _____

Student ID: _____

This exam is closed book and notes. 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 140 minutes to complete your exam. Be wise in managing your time. Good luck.

Please write the answer in English. If you are hand-writing, please make sure that it is legible (e.g., do not use cursive when writing the code).

Question 1	/ 25
Question 2	/ 35
Question 3	/ 20
Question 4	/ 30
Question 5	/ 30
Question 6	/ 5
Extra	/ 5 points
Total	/145

Name: Student ID:

1. (25 points) Understanding C Programs

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

(1) (6 points) Take a look at the code snippet below.

```
unsigned int i;
for (i=10;i>=0;i--)
    printf("KAIST");
```

(1-a) What is the output of the code snippet?
KAIST is printed out indefinitely.

(1-b) Explain what happens briefly.
i>=0 is always true because i is an unsigned int.

(2) (6 points) Take a look at the code snippet below.

```
unsigned int i;
for (i=10;i>-1;i--)
    printf("KAIST");
```

(2-a) What is the output of the code snippet?
Nothing is printed out.

(2-b) Explain what happens briefly.
i>-1 is never true because -1 gets converted to a large unsigned int.
Grading criteria: If value of max unsigned int is wrong, no point is given

(3) (5 points) What's the output of this code snippet? (%zu prints a value of unsigned long int, %p prints the address of a pointer)

```
char a[] = "kaist";
char *b = "kaist";
int *c;
int (*d)[100];
int *e[100];

printf("1: %zu 2:%zu 3:%zu 4:%zu 5:%zu\n",
        sizeof(a), sizeof(b), sizeof(c), sizeof(d), sizeof(e));
```

Output: (1 point for each)

1: _____ 2: _____ 3:_____ 4:_____ 5:_____

Solution: 6, 8, 8, 8, 800

Name: Student ID:

(4) (6 points) Numbers and C

(4-a) What is the value of sx after the following statements?

```
unsigned char ux = 255;
char sx = ux;
```

Solution: -1

(4-b) What is the value of ux after the following statements?

```
char sx = -1;
unsigned char ux = sx;
```

Solution: 255

(4-c) The following code prints out "0xfffffffc" and c is 1100 in binary.

```
int x = -4;
printf ("0x%x", x);
```

What would the following code print out?

```
printf ("0x%x %d \n", x>>2, x>>2);
printf ("0x%x %d \n", x<<1, x<<1);
```

_____

_____

0xffffffff -1
0xfffffff8 -8

(5) (2 points) What is sizeof("a") and sizeof("\0")?

2 and 2

2. (35 points) Writing and Understanding String Functions

You are writing C runtime library functions for string manipulation and want to add more functions to the library.

(1) (20 points) Recursive strstr().

char * strstr(const char* haystack, const char* needle);

finds a substring, needle, in the string, haystack and returns the starting address of the substring (or NULL if there is no such substring).

Name: Student ID:

Please fill in the blank to complete the implementation of strstr().

```
char *strstr(const char *haystack, const char *needle)
{
    assert(haystack);
    assert(needle);

    if(*needle == 0)    // base case where needle reached the end of string
        return (char *)haystack;

    if(*haystack == 0)    // base case where haystack reached the end of string
        return NULL;

    if(_____A_____    &&    // Hint: first char matches at haystack
        strstr(haystack + 1, needle + 1) == haystack + 1)    // remaining sequence
matches
        return (char *)_____;    // needle found in haystack

    return strstr(_____, _____); // otherwise (the first char doesn't match)
}
```

This space is left blank on purpose
Each blank is worth 5 points (20 points total).

Name: Student ID:

(2) (15 points) What is the output of the following program?

```
#include <string.h> /* strspn() strcspn() */
#include <stdio.h>
char *strfunc(char * str, const char * delim)
{
    static char* p=0;
    if(str)
        p=str;  // remember str using p
    else if(!p)
        return 0; // str is not provided and p is null

    str=p+strspn(p,delim);
    p=str+strcspn(str,delim);
    if(p==str)
        return p=0;
    p = *p ? *p=0,p+1 : 0;
    return str;
}
int main()
{
    char *token;
    char str[] = "Korea:Portugal;Uruguay,Ghana";

    token = strfunc(str, ",;:");

    while(token!= NULL ) {
       printf( " %s\n", token);
       token = strfunc(NULL, ",;:");
    }
}
```

Note1: In *p=0,p+1, a comma operator is used. The comma operator (represented by the token ,) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).

Note2: Below explains strspn() and strcspn()

```
#include <string.h>

size_t strspn(char *str, const char *accept);
size_t strcspn(char *str, const char *reject);
```

`strspn()` will tell you the length of a string consisting entirely of the set of characters in *`accept`*. That is, it starts walking down *`str`* until it finds a character that is *not* in the set (that is, a character that is not to be accepted), and returns the length of the string so far.

Name: Student ID:

`strcspn()` works much the same way, except that it walks down *`str`* until it finds a character in the *`reject`* set (that is, a character that is to be rejected.) It then returns the length of the string so far.

Return Value

The length of the string consisting of all characters in *`accept`* (for **`strspn()`**), or the length of the string consisting of all characters except *`reject`* (for **`strcspn()`**

Example

```
Size_t n;
char str1[] = "a banana";
char str2[] = "the bolivian navy on maneuvers in the south pacific";

// how many letters in str1 until we reach something that's not a vowel?
n = strspn(str1, "aeiou");  // n == 1, just "a"

// how many letters in str1 until we reach something that's not a, b,
// or space?
n = strspn(str1, "ab "); // n == 4, "a ba"

// how many letters in str2 before we get a "y"?
n = strcspn(str2, "y"); // n = 16, "the bolivian nav"
```

Answer:

Korea
Portugal
Uruguay
Ghana

Name: Student ID:

3. (20 points) Quick Sort

```
void swap(int* a, int* b){ int t = *a; *a = *b; *b = t; }

/* Partition takes array, sets the pivot element to arr[high] and
rearranges an array such that items less than or equal to the
pivot stays at the left of the array. It returns the pivot
position */
int partition (int arr[], int low, int high)
{
  int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high- 1; j++) {
    if (arr[j] <= pivot) {
        i++;
        swap(&arr[i], &arr[j]); // smaller than pivot
    }
  }
  swap(&arr[i + 1], &arr[high]); // move pivot to arr[i+1]
  return (i + 1); // pivot location
}
```

Fill in the blank to complete the quicksort() function. Use the function(s) given above.

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)                    // (1)
    {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);                // (2)
    }
}
```

Grading criteria:

(1) When there is suitable conditional statement for 'low' and 'high': +5 points

(2-1) When partition() and quickSort() are implemented correctly: +15 points

(2-2) When there is a mistake like quickSort(arr, low, pivot) or quickSort(arr, pivot, high): +10 points

(2-3) When quickSort() is recursively applied after partition(): +5 points

(3) When sorting is implemented with loop statements such as for() or while(): -20 points

(4) Syntax error: -1 point per each (maximum -3 points)

Name: Student ID:

4. (30 points) Linked List

The below header file provides the interface of a table module, implemented as a linked list.

```
1  /* linkedlist.h */
2  typedef struct Node {
3     int value;
4     struct Node *next;
5  } Node;
6
7  typedef struct {
8      Node *first;
9  } Table;
10
11 Table *Table_create(void);
12
13 void Table_print(Table* t);
14 void Table_add(Table* t, const int value);
15 int Table_sum(Table* t);
```

(4-1) Write a recursive function that prints out the values in a table in the reverse order of the list (i.e., print out the last item first and first item last).

```
void Table_print(Table* t)
{
  /* the print format doesn't matter, but printf() should be called after
  Table_print() */
  Table new_t;
  if (t->first == NULL)
    return;
  new_t.first = t->first->next;
  Table_print(&new_t);
  fprintf(stdout, "%d\n", t->first->value);
}
```

(4-2) Write a recursive function that returns the sum of values in the table

```
int Table_sum(Table* t)
{
  Table new_t;
  if (t->first == NULL)
    return 0;

  new_t.first = t->first->next;

    return t->first->value + Table_sum(&new_t);

}
```

(4-3) Write an iterative function that returns the sum of values in the table

```
int Table_sum(Table* t)
{
  Node *p;
  int sum = 0;
  p = t->first;
  while (p) {
    sum += p->value;
    p = p->next;
  }
  return sum;
}
```

Name: Student ID:

4-1 Grading criteria

-4) If original data(`t`) is distorted

-2) If no NULL check for `t->first`

-1) Minor mistake (e.g., no semicolon)

4-2 Grading criteria

- NULL check for `t-> first`: +2pts
- moving on to next node: +4pts
- using recursive function and sum appropriately: +4pts
- changing the data by changing `t->first`: -4pts

4-3 Grading criteria

-5) If wrong overall structure of the loop

-2) If missing initialization

-2) If wrong boundary condition of the loop

-1) If missing semicolon at least one time

Name: Student ID:

5. (15 points) Debugging

table.h

```
#ifndef TABLE_H_
#define TABLE_H_
/* table.h */
typedef struct Table Table;

Table* Table_create(void);
void       Table_add(Table* t, const char *key, int value);
int          Table_search(Table* t, const char *key, int * value);
int          Table_remove(Table* t, const char *key); // remove a node whose key matches
unsigned int hash(const char *x);
#endif
```

Name: Student ID:

table.c

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

enum {BUCKET_COUNT = 1024};

struct Node {
  char *key;
  int value;
  struct Node *next;
};
struct Table {
  struct Node *array[BUCKET_COUNT];
};

unsigned int hash(const char *x)
/*omitted for brevity. Assume hash returns a value between 0 and BUCKET_COUNT-1 */

struct Table *Table_create(void) {
   struct Table *t;
   t = (struct Table*)malloc (sizeof(struct Table));
   return t;
}

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

Name: Student ID:

```
int Table_search(struct Table *t, const char *key, int *value)
{
   struct Node *p;
   int h = hash(key);
   for (p = t->array[h]; p != NULL; p = p->next)
      if (strcmp(p->key, key) == 0) {
         value = p->value;
         return 1;
      }
   return 0;
}
```

(5-1) There is a total of three bugs in table.c. Find and fix the bugs. (15 points)

(a) Table_create(): You need to calloc() instead of malloc()

(b) Table_add(): You need to perform malloc or strdup and assign it to p->key

(c) Table_search(): Change “value = p->value” to “*value = p->value”

Grading criteria for 5-1)

- Applies identically for (a), (b), and (c)

1 point:

If you find the bug with the correct reason but fix it wrong.

3 or 4 points:

If you find the bug and fix it correctly but it has some minor mistakes (e.g. grammar)

5 points:

If you find the bug and fix it correctly.

Name: Student ID:

(5-2) Choose all the code snippet that is problematic (i.e., that has a bug). And explain why. (15 points)

char first[]=”good”;

char* second=”morning”;

(a)	strcat(first, second);
(b)	strcat(second, first);
(c)	strlen(strcat(first, “a”));

Grading criteria for 5-2)

+2 for correct answer

+3 for correct reasoning

(a), (c) -> strcat behavior is undefined if the destination array is not large enough for the contents of both src and dest and the terminating null character

(b) -> you should not use a literal string for a destination

Name: Student ID:

6. True/false questions (5 points)

   Mark true or false for each statement below.

   (1) In general, a module that has a large number of interfaces is better than ones with a few interface functions. (T/F)

   (2) When you design a module, you want to expose the implementation to the client to let the client easily understand how the module is written. (T/F)

   (3) Using a global variable to report an error to the client is not a good option for multi-threaded programs. (T/F)

   (4) assert() does not take any effect if NDEBUG macro is defined. (T/F)

   (5) A static local variable has automatic storage duration. (T/F)

   F, F, T, T, F

Name: Student ID:

(Extra 5 points) Do not write your name on this sheet of paper.

What do you like about this course? 5 points for answering this question in a meaningful way.