Spring Semester 2021 EE209 Mid-term Exam

Pledging of No Cheating

Note: Please write down your student ID and name, sign on it (draw your signature), and date it. If you do not fill out this page, we won't grade your mid-term exam.

저는 이번 시험을 온라인으로 치르면서 이 과목에서 금지한 어떤 부정행위도 저지르지 않을 것임을 서약합니다. 추후에 위반사항이 발견되었을 경우 합당한 모든 불이익을 감수하겠습니다.

I pledge that I will not participate in any activity of cheating disallowed by this course while taking this exam online. I will assume full responsibility if any violation is found later.

Student ID: _____ Name: _____

Signature: _____ Date: _____

Name: Student ID:

Spring Semester 2021

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 120 minutes (1:10 PM – 3:10 PM) to complete your exam. You can submit your answers early but you can leave the zoom session from 3:00PM. Good luck.

Question 1	/ 10
Question 2	/ 20
Question 3	/ 15
Question 4	/ 5
Question 5	/ 20
Question 6	/ 10
Question 7	/ 20
Total	/ 100

Name: Student ID:

Assume all programs in this exam run on 64-bit Linux (x86-64). That is, assume that the code runs on eelab6.kaist.ac.kr

1. **(10 points) Numbers & Strings**

(a) **(2 points)** what's the output of the following code?

```
1  #include <stdio.h>
2  int main( ) {
3    char x = -1; printf ("x =%d₩n", (int)(unsigned char)x); return 0;
4    }
```

Your answer: x = ____255_____

(b) **(6 points)** what's the output of the following code? (1 point each)

```
1  #include <stdio.h>
2  #include <string.h>
3  int main( ) {
4    char *p ="abc";
5    char a[] ="abc";
6    printf ("sizeof(p) = %zu₩n", sizeof(p));
7    printf ("sizeof(*p) =%zu₩n", sizeof(*p));
8    printf ("sizeof(a) = %zu₩n", sizeof(a));
9    printf ("sizeof(*a) = %zu₩n", sizeof(*a));
10   printf ("sizeof(&*a) = %zu₩n", sizeof(&*a));
11   printf ("strlen(p) = %d₩n", strlen(p));
12   return 0;   }
```

Line 6: sizeof(p) = _____8_____

Line 7: sizeof(*p) = ____1_____

Line 8: sizeof(a) = _____4_____

Line 9: sizeof(*a) = ____1_____

Line 10: sizeof(&*a) = ___8_____

Line 11: strlen(p) = _____3_____

Name: Student ID:

(c) **(2 points)** How many times is Line 7 executed?

```
1  #include <stdio.h>
2  #define NELEMS(x) (sizeof(x)/sizeof(x[0]))
3  int main( ) {
4    int i;
5    int y[10];
6    for (i = 10; i - NELEMS(y) >= 0; i-= 10)
7        printf("current i is %d₩n", i);
8    return 0;
10   }
```

Your answer: ____infinite number – infinite loop, the code never gets out the loop____

Name: Student ID:

2. **(20 points) Data Types, Functions & Pointers**

Consider the following C program. **(a)/(b)**

```
1  #include <stdio.h>
2  int main( ) {
3        char ch;
4        while ((ch = getchar()) != EOF) putchar(ch);
5        printf("OK I'm ready to stop₩n");
6        return 0;
7  }
```

(a) (2 points) I run the program without any command line arguments, and then type Ctrl+D right away (which would end stdin). Which one of the followings is correct?
(1) The program prints out the message in line 5 and stops
(2) The program never reaches line 5, so it does not stop

Your answer: ____(1)____

(b) (3 points) I changed the type of ch to unsigned char (to unsigned char ch; in line 3) and I compiled the code, and repeat what I did for (a) -- run the program without any command line arguments, and then type Ctrl+D right away. Which one of the followings is correct?
(1) The program prints out the message in line 5 and stops
(2) The program never reach line 5 and does not stop.

Your answer: ____(2)____

(c) (5 points) I compiled the code below, and ran it. I typed in 345 and an enter ('\n'). Then, I see "x's address is 0x7fffe66ff82c" on my monitor. What's the output from line 7?

```
1  #include <stdio.h>
2  int main() {
3     int x;
4     unsigned long int y = (long int)&x;
5     scanf("%d", (int *)y);
6     printf("x's address is %p₩n", &x);
7     printf("x = %d y = %lx₩n", x, y);     // %lx prints the argument as hexadecimal format
8     return 0;
9  }
```

Name: Student ID:

Your answer: : ____x = 345__y=0x7fffe66ff82c____

(d) (5 points) What's the output of the following program? Information: Intel CPU is a little endian processor where the least significant byte of an integral type (e.g., short, int, long) is placed at the lowest memory address. For example, let's say int x; and x's address is 0x1234. Then x's least significant byte is at 0x1234 and the next byte is at 0x1235, and the most significant byte is at 0x1237.

```
1  #include <stdio.h>
2  union X { int x; unsigned char y;};
3  int main() {
4     union X a;
5     int *p = &a.x;
6     printf("p is %p₩n", p);
7     printf("a.y's address is %p₩n", &a.y);
8     a.x = -1;
9     printf("1: a.x =%d a.y =%d₩n", a.x, a.y);
10    a.y = 0xfe;
11    printf("2: a.x =%d a.y =%d₩n", a.x, a.y);
12    return 0;
   }
```

At line 6, the program prints out "p is 0x7fffea930f0c". Fill out the blanks.

Line 7: a.y's address is _____0x7fffea930f0c_____

Line 9: 1: a.x =____-1_____ a.y = _____255_____

Line 11: 2: a.x =__-2_____ a.y = _____254_____

Name: Student ID:

(e) (5 points) Line 10 in the code below prints out "str is = 0x7fb869200807 arr is 0x7ffffb7c8e4d" to the monitor.

```
1   #include <stdio.h>
2   void f(char a[], char *p) {
3       printf("a is = %p p is %p₩n", a, p);
4       a[0] = 'A';
5       *p = 'A';
6   }
7   int main() {
8     char *str = "Hi EE209A";
9     char arr[] = "Hi EEE209B";
10    printf("str is = %p arr is %p₩n", str, arr);
11    f(str, arr);
12    return 0;
    }
```

(e-1) What's the output in Line 3? **(2 points)**

Your answer: a is ____0x7fb869200807___ p is___0x7ffffb7c8e4d_____

(e-2) The program crashes when it runs. At which line does it crash **(1 point)**? Explain why it crashes **(2 points).**

Your answer: The program crashes at Line _____4_____

The It crashes because __it tries to modify/update the value of a string constant (or it tries to update the memory content at read-only memory section)_____

Name: Student ID:

3. **DELETED**

Name: Student ID:

4. **(5 points)** Write void reverse(char *s); reverse() gets an input string as an argument, and reverses the string. For example, if s = “abcde”, reverse(s); printf(“s=%s”, s); then printf() would print out “edcba”. Fill in the blank in the code below.

```
1  void reverse(char *s)
2  {
3      char *e, temp;
4      assert(s != NULL);
5
6      e = s + strlen(s) -1; /* e points to the last character in the string */
7      while (s < e) {
8        /* swap *s and *e, increment s by 1, decrement e by 1 */
9              (1)
10     }
11 }
```

Your answer:

temp =*s;*s =*e; *e =temp; s++, e--;

Name: Student ID:

5. **(20 points) Recursion**

(a) (5 points) Fun function – consider the following code snippet.

```
1  int fun(int x, int y)
2  {
3     If (x == 0) return y;
4     return fun(x-1, x+y);
5  }
```

(a-1) **(2 points)** what does fun(3, 2) return?

Your answer: _____8_____

(a-2) **(3 points)** Briefly explain what this function calculates in terms of x and y.

Your answer: _____it calculates x(x+1)/2+y or (1+2+...+x)_____

Name: Student ID:

(b) (15 points) PrintAll(int x) prints all possible binary numbers with the length, x. For example, PrintAll(2) prints out four binary numbers – 00, 01, 10, 11. We implement this function with recursion. The key idea is to define PrintWithPrefix(int x, char prefix[], int len) where x is the original length, prefix is the binary prefix that has been constructed so far, and len is the length of the current prefix. If len becomes x, the function has finished constructed ONE binary number whose length is x. So, it prints out the prefix.

```
1  void PrintWithPrefix(int x, char prefix[], int len)
2  {
3      /* len is the length of prefix, if len == x, we need to print it out */
4      if (len == x) {
5          prefix[len] = 0;
           printf("%s₩n", prefix);
           return;
       }
       /* add '0' to the current prefix,
            and print all binary numbers with the newly constructed prefix */
       prefix[len] ='0';
       PrintWithPrefix(x, prefix,  (1)  );

       /* add '1' (instead of '0') to the current prefix, and do the same thing */
       (2)
   }

   void PrintAll(int x)
   {
      char buf[x+1];          // dynamically allocates x + 1 bytes to buf
      PrintWithPrefix(x, buf, 0);
   }
```

(b-1) **(5 points)** What should be the parameter that goes into (1)?

Your answer: _____len+1_____

(b-1) **(10 points)** Write the code for (2). Hint: Two lines should be enough.

Your answer:

```
prefix[len] ='1';
printWithPrefix(x, prefix, len+1);
```

Name: Student ID:

6. **(10 points) Dynamic Memory Allocation**

Consider the following code snippet. Assume # of input from stdin is smaller than 1 millions

```
1  int size = 2, i = 0;
2  int *p = malloc(size * sizeof(int)); // assume malloc() is successful
3  while (scanf("%d",   p + i) != EOF) {
4      if (i++ == size) {
5          size *= 2;
6          realloc(p, size * sizeof(int));       // assume realloc() is always successful
7       }
8   }
```

The code above crashes at line 3. Fix two lines in the code above to avoid the crash. Explain why original lines are wrong in each line. **(5 points each)**

Your answer:

1. Line 4 should be if (i++ == size -1) or if (++i == size). Why? i++ increments i after the line, so i holds the last index which is (# of integers read so far) - 1.
2. Line 6 should be p = realloc(p, size); if realloc() never fails. Why? The expanded memory by realloc() could be different from the original location (p). So, p should be updated to point to the new location. But in practice, realloc() can return NULL, so a safe way something like
   int *k = realloc(p, size);
   if (k != NULL) { p = k; }
   else { error processing; }

Name: Student ID:

7. **(20 points) Key-value store with a hash table**

The below header file provides the interface of a hash table for a key-value store

```
1  /* table.h */
2  Table *Table_create(void);
3  void Table_add(Table* t, const char *key, int value);
4  int Table_search(Table* t, const char *key, int * value);
5  int Table_remove(Table* t, const char *key); // remove a node whose key matches
```

The code below is the implementation of the hash table. Assume malloc() and calloc() are always successful – but in practice you must check if the return value is NULL.

```
1  /* table.c */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "table.h"
5
6  enum {BUCKET_COUNT = 1024};
7  struct Node {
8      const char *key;
9      int value;
10     struct Node *next;
11 };
12
13 struct Table {
14     struct Node *array[BUCKET_COUNT];
15 };
16
17 unsigned int hash(const char *x) {
18     int i;
19     unsigned int h = 0U;
20     for (i=0; x[i]!='₩0'; i++)
21         h = h * 65599 + (unsigned char)x[i];
22     return h % BUCKET_COUNT;
23 }
24
25 struct Table *Table_create(void) {
26     struct Table *t;
```

Name: Student ID:

```
27      t = (struct Table*)calloc(1, sizeof(struct Table));
28      return t;
29  }
30
31  void Table_add(struct Table *t, const char *key, int value)
32  {
33      struct Node *p = (struct Node*)malloc(sizeof(struct Node));
34      int h = hash(key);
35      p->key = key;
36      p->value = value;
37      p->next = t->array[h];
38      t->array[h] = p;
39  }
40
41  int Table_search(struct Table *t, const char *key, int *value)
42  {
43      struct Node *p;
44      int h = hash(key);
45      for (p = t->array[h]; p != NULL; p = p->next)
46          if (strcmp(p->key, key) == 0) {
47              *value = p->value;
48              return 1;
49          }
50      return 0;
    }
```

(a) **(10 points)** table.c is incomplete as Table_remove() is missing. Please fill in the function below. It removes the node whose key matches the parameter key. It returns 1 if it removed the node, 0 if a node with the matching key does not exist.

Your answer: one possible solution is

Name: Student ID:

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

Name: Student ID:

(b) The following code is a client

```
1  /* client.c */
2  #include <string.h>
3  #include "table.h"
4  int main ( ) {
5    char k[10] = "EE209A"
6    int value =0, res;
7    Table* t = Table_create(); // assume it's successful.
8    Table_add(t, k, 10);
9    strcpy(k, "EE209B");
10   res = Table_remove(t, "EE209A");
11   printf("Table_remove() returns %d₩n", res);
12   res = Table_search(t, "EE209B", &value);
13   printf("Table_search() returns %d and value is %d₩n", res, value);
14   return 0;
15 }
```

Assume hash("EE209A") != hash("EE209B")

(b-1) **(5 points)** What does Line 11 print out? Explain why Table_remove() returns such a value?

Your answer; ___Table_remove() returns 0. Why? As it cannot find the node with "EE209A" as its key since the keyt is changed from "EE209A" to "EE209B" in Line 9_____

(b-2) **(5 points)** What does Line 13 print out? Explain

Your answer; ___Table_search() returns 0 and value is 0. Why? As it cannot find the node with "EE209B" as its key since hash("EE209B") != hash("EE209A"), so it looks the node in the different bucket than the one Line 8 added the node to._____