Spring Semester 2025 EE209 Midterm 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 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: _____

The 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. All your answers must be included in the attached sheets. You have 150 minutes to complete your exam. Be wise in managing your time.

Please do not fill in the "Score" fields below. Self-grading is not allowed. Good luck!

1	/15
2	/15
3	/8
4	/8
5	/15
6	/15
7	/13
8	/15
Total	/104

1. (15 points) C fundamentals and data types

(a) (3 points) Consider the following three integers given in different ways of representation in C:

a = 0b101101

b = 075

c = 0x3F

Write down a, b, c in decimal form.

[Solution]

a = 45, b = 61, c = 63

(b) (4 points) In an 8-bit two's complement system, for the following numbers, convert decimal ones into binary and binary ones into decimal representation:

a. -18

b. -128

c. 00110100

d. 10101010

[Solution] 11101110, 10000000, 52, -86

(c) (4 points) Write down the printing output of the following C code snippet:

```
#include <stdio.h>
int main() {
    unsigned char ch = - 1;
    printf("a = %d, b = %o, c= %x\n", ch,
ch, ch);
    printf("sizeof(unsigned char) = %zu
byte(s)", sizeof(unsigned char));
    return 0;
}
```

[Solution]

a = 255, b = 377, c = ff,

sizeof(unsigned char) = 1 byte(s)

(d) (3 points) Consider the following C code snippet. Write down the exact output that the program would print when executed. Assume all variables are 8-bit signed integers and that all operations are performed using 8-bit signed arithmetic.

Hint: Operators follow the precedence rule in the following order: additive(higher) > bitwise > logical(lower)

```
#include <stdio.h>
int main() {
    int a = 3, b = 5;
    int x = (a < b) && (b << 1) + 1;
   int y = ((x + 2) & b) ^ (~b);
   int z = x | b && y;
   printf("x, y, z = %d, %d, %d\n", x,
y, z);
   return 0;
}
```

[Solution] x, y, z = 1, -5, 1

(e) (1 point) Write down the printing output of the following C code snippet.

```
#include <stdio.h>
int main() {
    int a = 2, b = 4, c =5;
   int d = a++ * --b + ++c - a * b--;
   printf("d = %d\n", d++);

   return 0;
}
```

[Solution] d=3

2. (15 points) Functions

(a) (6 points) Fill in the blank. Complete `ReverseOfString` function that reverses a string using recursion. The function should modify the string in-place and does not return any value. It takes three arguments: the string, the index of leftmost character, and the index of rightmost chracter. For example, given the string EE209, calling `ReverseOfString(EE209,0,4)` should result in the reversed string 902EE.

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

void ReverseOfString (char *str, int left, int right) {

if (left >= right)
        return;

    char temp = str[left];
    str[left] = str[right];
    str[right] = temp;

    ReverseOfString ( (a), (b), (c) );
}
```

[Criteria]

(a) (2p) str

(b) (4p) left + 1

(c) (4p) right -1

(b) (9 points) Fill in the blank. A symmetrical binary number is an integer whose binary representation reads the same backward as forward. Complete `IsBinarySymmetrical` function that returns 1 if the given integer is symmetrical in its binary form and 0 otherwise. For example, 5 is a symmetrical binary number, because its binary representation is 101. In contrast, 6 is not symmetrical because its binary representation, 110, is not the same when reversed. Assume that x > 0.

```
#include <stdio.h>

int IsBinarySymmetrical(int x) {

    int numBits = 0;
    int temp = x;

    while (temp > 0) {
        numBits++;
        temp = __(a)__;
    }

for (int i = 0; i < numBits / 2; i++) {
        int leftBit  = __(b)___;
        int rightBit = __(c)___;

        if (leftBit != rightBit)
            return 0;
}
    return 1;
}
```

[Criteria]

(a) (3p) temp / 2

(b) (3p) (x >> i) & 1

(c) (3p) (x >> numBits - 1 – i) & 1

3. (8 points) Consider the following code written in C. Assume a 64-bit Linux system (x86-64). For each of the 8 `printf` statements (commented Line 1 to Line 8), write down the value, character, or string that is printed. (+1 point for each correct answer, 8 points maximum)

```
#include <stdio.h>

int main(void) {

    char str[] = "KAIST EE209";
    char *p = str;
    char *q = p + 6;
    char **r = &p;

    printf("%c\n", *(p + 3));      // Line 1
    printf("%c\n", q[-2]);         // Line 2
    printf("%c\n", **r);           // Line 3
    printf("%zu\n", sizeof(str)); // Line 4

    *p = 'k';
    p += 2;
    p[3] = 'S';

    printf("%s\n", str);           // Line 5

    char *u = str;
    u++;

    printf("%c\n", *(++u));        // Line 6
    printf("%c\n", (*(--u))++);    // Line 7
    printf("%s\n", --u);           // Line 8

    return 0;
}
```

[Criteria] (+1 point for each correct answer)

Line 1: S

Line 2: T

Line 3: K

Line 4: 12

Line 5: kAISTSEE209

Line 6: I

Line 7: A

Line 8: kBISTSEE209

4. (8 points) Consider the following code written in C. For each of the 4 printf statements (commented strcmp problem 1 to 4), answer whether the resulting number is positive, negative, or zero. (+2 points for each correct answer, **-1 point for each wrong answer**)

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

char* mystery_function(char *s, int magical_number) {
    for (int i = 0; s[i] != '\0'; i++) {
        // check whether the character is an alphabet
        if (isalpha(s[i])) {
            // check whether the character is a lowercase or
uppercase letter
            if (islower(s[i]))
                s[i] = ((s[i] - 'a' + magical_number) % 26) + 'a';
            else if (isupper(s[i]))
                s[i] = ((s[i] - 'A' + magical_number) % 26) + 'A';
        }
    }
    return s;
}

int main(void) {
    char str1[] = "one"; char str2[] = "bar";
    char str3[] = "CAT"; char str4[] = "PNG";

    mystery_function(str1, 13);
    mystery_function(str4, 13);

    printf("%d\n", strcmp(str1, str2)); // strcmp problem 1
    printf("%d\n", strcmp(str2, str3)); // strcmp problem 2
    printf("%d\n", strcmp(str3, str4)); // strcmp problem 3
    printf("%d\n", strcmp(str4, str1)); // strcmp problem 4

    return 0;
}
```

Answer:

Strcmp problem 1: 0 (zero)

Strcmp problem 2: + (positive)

Strcmp problem 3: 0 (zero)

Strcmp problem 4: - (negative)

5. (15 points) Structures and dynamic memory management

(a) (2 points) Fill in the Blanks

Fill in each blank with the correct term:

In C, the function (i)_____ allocates memory without initializing it, while the function (ii)_____ allocates memory and sets all bytes to zero. In addition, a(an) (iii)_____ is a user-defined type that groups variables of different types, whereas a(an) (iv)_____ stores a collection of elements of the same type.

Answer:

(i) malloc

(ii) calloc

(iii) structure

(iv) array

(b) (3 points) Short Answer (Union vs. Structure)

Consider a structure and a union that both contain an int and a double. Explain the difference in their memory allocation and state which one typically requires less memory.

Answer:

In a structure, separate memory is allocated for each member so its size is approximately sizeof(int) + sizeof(double) (plus any padding). In a union, all members share the same memory area, and its size is that of the largest member (typically sizeof(double)). Therefore, the union requires less memory because it allocates only enough space for its largest member.

(c) (4 points) – Bug Identification

Examine the following code snippet:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    char *buffer = malloc(10);
    if (buffer == NULL) {
        return 1;
    }
    strcpy(buffer, "HelloWorld!");
    printf("Buffer: %s\n", buffer);
    free(buffer);
    printf("Buffer: %s\n", buffer);
    return 0;
}
```

Identify and briefly explain the **two** bugs in the code, and suggest corrections.

Answer:

1. Buffer Overflow: The string "HelloWorld!" needs 12 bytes (11 characters plus the null terminator) but only 10 bytes are allocated.
   Correction: Allocate memory using malloc(strlen("HelloWorld!") + 1).
2. Dangling Pointer Usage: After free(buffer), the pointer buffer is used in the subsequent printf, which is undefined behavior because it becomes a dangling pointer.
   Correction: Do not access buffer after freeing it (or set buffer to NULL immediately after free(buffer)).

(d) (6 points) Code Output with Complex Structure and Dynamic Memory

Examine the following code snippet and determine the output when `main()` is executed.

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

typedef struct {
    char *data; int id;
} Record;

int my_strlen(const char *s) {
    int i = 0;
    while (s[i])
        i++;
    return i;
}

int main(){
    Record *r1 = malloc(sizeof(Record)), *r2 = malloc(sizeof(Record));
    r1->data = malloc(6); r2->data = malloc(6);
    char alpha[] = "Alpha", beta[] = "Beta";
    for (int i = 0; i < sizeof(alpha); i++) r1->data[i] = alpha[i];
    for (int i = 0; i < sizeof(beta); i++) r2->data[i] = beta[i];
    r1->id = 12; r2->id = 34;
    *(r1->data + 2) = *(r2->data + 1);

    int len = my_strlen(r2->data);
    char *newData = malloc(len+2);
    for (int i = 0; i < len; i++)
        newData[i] = r2->data[i];
    newData[len] = '0' + (r1->id % 10);
    newData[len+1] = '\0';
    free(r2->data);
    r2->data = newData;

    printf("%s(%d) %s(%d) \n", r1->data, r1->id, r2->data, r2->id);
    free(r1->data); free(r2->data); free(r1); free(r2);
    return 0;
}
```

Note: In ASCII, the digits '0' through '9' are stored consecutively. The character '0' has an integer value of 48. Thus, to convert an integer (0–9) to its corresponding character, you add '0' to the integer.

Answer:

Aleha(12) Beta2(34)

6. (15 points) Linked Lists

We implement a singly linked list and two functions for modifying the list. Based on the below structure and functions, answer the following questions. A function `insertNode(struct Node **head, int value)` inserts a new node at the head.

```
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
};

void mysteryA(struct Node **head) {
    struct Node *prev = NULL, *curr = *head, *next;
    while (curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}

void mysteryB(struct Node **head, int i, int j) {
    struct Node *prev_i = NULL, *curr_i = *head;
    struct Node *prev_j = NULL, *curr_j = *head;

    for (int pos = 0; curr_i != NULL && pos < i; pos++) {
        prev_i = curr_i;
        curr_i = curr_i->next;
    }
    for (int pos = 0; curr_j != NULL && pos < j; pos++) {
        prev_j = curr_j;
        curr_j = curr_j->next;
    }

    if (prev_i) prev_i->next = curr_j;
    else *head = curr_j;
    if (prev_j) prev_j->next = curr_i;
    else *head = curr_i;

    struct Node* temp = curr_i->next;
    curr_i->next = curr_j->next;
    curr_j->next = temp;
}
```

(a) (4 points) Write the final sequence of node values (from head to tail) after this operation on an empty list?

```
struct Node *list1 = NULL;
insertNode(&list1, 60);
insertNode(&list1, 50);
insertNode(&list1, 40);
insertNode(&list1, 30);
insertNode(&list1, 20);
insertNode(&list1, 10);
mysteryA(&list1);
```

[Answer]

60, 50, 40, 30, 20, 10

(mysteryA: reverse entire list order)

[Criteria]

- No partial point

(b) (4 points) Write the final sequence of node values (from head to bottom) after this operation on an empty list?

```
struct Node *list2 = NULL;
insertNode(&list2, 10);
insertNode(&list2, 20);
insertNode(&list2, 30);
insertNode(&list2, 40);
insertNode(&list2, 50);
mysteryA(&list2);
insertNode(&list2, 60);
mysteryB(&list2, 2, 4);
```

[Answer]

60, 10, 40, 30, 20, 50

(mysteryB: swap two node at indices i and j)

[Criteria]

- No partial point

(c) (7 points) Fill in the blank to complete the following function, which inserts all nodes of `list1` into `list2` at the specified position `i` (with the `head` of `list2` considered as position 0).

```
// Insert list1 into list2 at position i
void insertList(struct Node **list2, struct Node *list1, int i)
{
    if (i <= 0) {
         struct Node *tail = list1;
         while(tail->next != NULL)
             tail = tail->next;

         ____________________________;              /* [Blank 1] */
         *list2 = list1;
         return;
    }
    struct Node *curr = *list2;
    int pos = 0;
    while (curr != NULL && pos < i - 1) {
         curr = curr->next; pos++;
    }
    if (!curr) return;
    struct Node *temp = curr->next;

    ____________________________;                  /* [Blank 2] */

    struct Node *tail = list1;
    while(tail->next != NULL)
         tail = tail->next;

    ____________________________;                  /* [Blank 3] */
}
```

Example)

```
struct Node *list3 = NULL;
insertNode (&list3, 30);
insertNode (&list3, 20);
insertNode (&list3, 10);
struct Node *list4 = NULL;
insertNode (&list4, 500);
insertNode (&list4, 400);
insertNode (&list4, 300);
insertNode (&list4, 200);
insertNode (&list4, 100);
insertList(&list4, list3, 3);
```

Then, it should result in a combined list (list 4) of 100, 200, 300, 10, 20, 30, 400, 500.

[Answer]

Blank 1: tail->next = *list2

Blank 2: curr->next = list1

Blank 3: tail->next = temp

[Criteria]

- Each wrong answer (-2)
- No answer (-7)

7. (13 points) Hash table

(a) (9 points) State whether the following statements about hash table are true or false. If the statement is true, write **True.** If not, write **False** (+1 points for each correct answer, -1 points for each wrong answer. 0 points for no answer)

1) Hash function is an injective function.

Your answer: _____ False. Collision can happen.

2) A size of range of hash function is lower than array size.

Your answer: _____ False Generally, a size of range of hash function is same as ARRAYSIZE.

3) Hash table allows float type keys .

Your answer: _____ True

4) When you insert N different keys into a hash table of size m, the minimum number of keys in the same bucket is N/m.

Your answer: _____ False 0.

5) Dictionary is an ADT specialized for random access.

Your answer: _____ True

6) Linked list is a data structure for implementing hash table.

Your answer: _____ False Linked list is not necessary for hash table.

7) Hash collisions can be completely avoided by using a strong hash function.

Your answer: _____ False It depends on the distribution of keys.

8) Time complexity for random access in hash table is not always O(1).

Your answer: _____ True

9) Hash table which is implemented with linked list can restore the order of input sequence.

Your answer: _____ False Hash function is time-invariant.

(b) (2 points) What is the output produced by the following code when inserting "EE"?

Hint: 'E' has an ASCII code of 69 (decimal)

```
unsigned int hash(const char *x) {
int i;
unsigned int h = 0U;
for (i=0; x[i] != '\0'; i++)
    h = h * 65599 + (unsigned char)x[i];
return h % 1024;
}
```

Your answer: _____

320

(c) (2 points) State 2 reasons why the following code is bad.

```
unsigned int bad_hash(const char* key) {
    return strlen(key);
}
```

Reason 1: _____

Reason 2: _____

(Over array size)

A range of the function is not bounded.

(Many collisions. If the answer doesn't have the word collision, 0 points. E.g. High time complexity for random search)

The number of collisions will greatly increase

8. (15 points) Binary Search Tree

Let us write a C program that implements a binary search tree and its add and remove operation. Below is the basic code provided for you.

```
struct Node {
    const char *key;
    int value;
    struct Node *left;
    struct Node *right;
};
struct Tree {
    struct Node *root;
};
struct Tree *Tree_create(void) {
    struct Tree *t;
    t = (struct Tree *)calloc(1, sizeof(struct Tree));
    return t;
}
```

(a)~(c) Fill in the blanks to implement add operation of your binary search tree. The answer should be composed of one single code statement. Points will be deducted for syntax errors.

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

void Tree_add(struct Tree *t, const char *key, int value) {
    struct Node *p = (struct Node *)malloc(sizeof(struct
Node));
    p->key = key;
    p->value = value;
    p->left = ___(a)___;
    p->right = ___(a)___;
    struct Node **cur = &t->root;
    while (*cur != NULL){
        if (strcmp(key, (*cur)->key < 0)) cur = &(*cur)->left;
```

```
        else if (strcmp(key, (*cur)->key > 0)) cur = &(*cur)-
>right;
        else {
            ____(b)____;
            ____(c)____;
            return;
        }
    }
    *cur = p;
}
```

(a) (2 points) Your answer: _____

(b) (3 points) Your answer: _____

(c) (3 points) Your answer: _____

Solution (No partial point)

(a) NULL

(b) (*cur)->value = value (or other equivalent single line code)

(c) free(p)

(d) (3 points) Implement find_max(struct Node* node) function that returns the maximum Node in the tree which has the input node as the root. The answer should be composed of one code block, which consists of one or more code statements. Points will be deducted for syntax errors.

```
struct Node *find_max (struct Node *node) {
   if (node == NULL) return NULL;
   // (d) Implement your code here




    return node;
}
```

Solution (example)

while (node->right != NULL) node = node->right;

(-1pt) minor syntax error

(e) (4 points) Implement remove operation of your binary search tree, by filling in the missing part below. The answer should be composed of one code block, which consists of multiple code lines. Use find_max function that you implemented in (d). Points will be deducted for syntax errors.

```
struct Node* Tree_remove_node (struct Node *root, int key) {
    if (root == NULL) return NULL;

    if (key < root->key) root->left = Tree_remove_node(root-
>left, key);
    else if (key > root->key) root->right =
Tree_remove_node(root->right, key);
    else {
        if (root->left == NULL && root->right == NULL) {
            free(root);
            return NULL;
        }
        else if (root->left == NULL) {
            struct Node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct Node *temp = root->left;
            free(root);
            return temp;
        }
        else {
            // (e) Implement your code here









        }
```

```
    }
    return root;
}

void Tree_remove (struct Tree *t, int key) {
    t->root = Tree_remove_node(t->root, key);
}
```

Solution (example)

struct Node *predecessor = find_max(root->left);

root->key = predecessor->key;

root->value = predecessor->value;

root->left = Tree_remove_node(root->left, predecessor->key);

(-1pt) minor syntax error

(-2pt) Updating only one of key or value or left of root Node, 2pts deduction per missing components.

(+1pt) find predecessor correctly.