Spring Semester 2023

EE209: Programming Structures for Electrical Engineering

Mid-term Exam

Closed book/notes/friends/electronic devices/everything.
Write everything in English.
Write your student ID and name on *every* page.
Hand it in by 3:50 PM
PLEASE MAKE YOUR HANDWRITING LEGIBLE.

Code of Conduct

- All coats and jackets should be placed on the back of each candidate's chair. All notes and books, pencil cases, turned-off cell phones, laptops, and other unauthorized aids, and purses should be stored inside the candidate's knapsack or large bag, which should then be closed securely and placed under the candidate's chair. Candidates are NOT allowed to have a pencil case on their desk; any pencil cases found on desks will be searched. All watches and timepieces on desks will be checked. Candidates are not allowed to touch their knapsack or bag or the contents until the exam is over. Candidates are not allowed to reach into the pockets or any part of their coat or jacket until the exam is over.
- Candidates shall not communicate with one another in any manner whatsoever during the examination. Candidates must stay in the examination room unescorted for any reason, including using the washroom.
- No materials or electronic devices shall be used or viewed during an examination except those authorized by the Chief Presiding Officer or Examiner. Unauthorized materials include, but are not limited to: books, class notes, or aid sheets. Unauthorized electronic devices include, but are not limited to, cellular telephones, laptop computers, tablets, calculators, MP3 players (such as an iPod), Personal Digital Assistants ("PDA" such as a Palm Pilot or Blackberry), electronic dictionaries, Smart Watches and Smart Glasses.
- Candidates who use or view any unauthorized materials or electronic devices while their examination is in progress - or who assist or obtain assistance from other candidates or any unauthorized source – will get the grade F and face possible suspension.
- At the conclusion of an examination, all writing shall cease. The Chief Presiding Officer may seize the papers of candidates who fail to observe this requirement, and a penalty may be imposed.

I have read and adhere to the code of conduct. My signature on this page means my pledge to abide by these standards.

Signature: _____ Date: _____

Name: _____

Student ID: _____

Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help in understanding the questions in case you find them unclear. Be concise and precise in your answers, and state clearly any assumption you may have made. You have 165 minutes (1:00 PM – 3:45 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 30
Question 2	/ 10
Question 3	/ 20
Question 4	/ 20
Question 5	/ 15
Question 6	/ 25
Extra	5 points

Total / 120

SID:	Name:

1. (30 points) Quick Hits

Please read the questions carefully. You get +2 points for each question if you answer the question correctly. If you do not answer, you get 0 points. If you answer the question incorrectly, you get -2 points.

(1) ~1 && 1
   a. True
   b. False

(2) 16 >> 4
   a. True
   b. False

(3) 0x2B | (! 0x2B)
   a. True
   b. False

(4) sizeof(5) > sizeof(2L)
   a. True
   b. False

(5) Convert the decimal number -9 into 8-bit 2's complement number. 11110111

(6) $(4000000000)_{10}$ can be represented as int in C.
   a. True
   b. False

(7) Which of the following is the output generated by the `printf` statement in the code segment below?

```
int x = 3;
int *y;
int *z;

y = &x;
z = y;
(*z)--;

printf("result: %d\n", *y + *z);
```

   a. result: 6
   b. result: 5
   c. result: 4
   d. None of the above.

SID:	Name:

(8) Which of the following statements regarding pointer variables is FALSE?

a. Working with an uninitialized pointer variable is a compile-time error.
b. The asterisk character (*) is used in two different contexts for pointer variables; for declaration and dereferencing.
c. The value of a pointer variable can change during the execution of a program.
d. None of the above.

(9) Suppose `x` is a double. After statements

```
x = 5.9;
a = (int) x;
```

have been executed, what is the value of `x`?

a. 5.9
b. 5
c. 5.0
d. 6

(10) What is the output of the `printf` statement in the code segment below?

```
int x = 0x15213F10 >> 4;
char y = (char) x;
unsigned char z = (unsigned char) x;
printf("%d, %u", y, z);
```

a. -241, 15
b. -15, 241
c. -241, 241
d. -15, 15

(11) Place parenthesis in the following expression to explicitly show the order of evaluation. For example, a + b * c ⇒ (a + (b * c))

3 + * p ++ ⇒ (3 + (* (p ++)))

(12) Which of the following statements regarding user-defined functions is FALSE?

a. A variable declared in the local declaration section of a function can have the same identifier as one of the parameters within the same function.
b. Data sent from the calling function to the function being called will be received in the same order in which it was passed.
c. Parameters are defined as local variables in the first line of the function definition and should not be re-defined within the local declaration section of the function.
d. None of the above.

SID:	Name:

(13) Which of the following describes the integer value generated by the `printf` statement in the code segment below?

```
int x[5] = {6, 9, 3, 0, 4};
int y[5] = {0};

y = x;

printf("y[0] = %d\n", y[0]);
```

a. The value displayed will be the integer 6.
b. The value displayed will be the memory address represented by the array x.
c. No integer value will be displayed due to a compiler error regarding the assignment statement.
d. None of the above.

(14) Which of the following is the correct ordering (left-to-right) of a file's compilation cycle (a filename with no extension is an executable)?

a. foo.c ⇒ foo.o ⇒ foo.s ⇒ foo
b. foo ⇒ foo.s ⇒ foo.o ⇒ foo.c
c. foo.c ⇒ foo.s ⇒ foo ⇒ foo.o
d. foo.c ⇒ foo.s ⇒ foo.o ⇒ foo

(15) From the following C declaration, what does `f` represent?

```
int *(*f[3])();
```

a. an array of pointers to pointers to functions that return int.
b. a pointer to an array of functions that return a pointer to int.
c. a function that returns a pointer to an array of pointers to int.
d. an array of pointers to functions that return a pointer to int.

SID:	Name:

2. (10 points) Linking

(1) (6 points) Consider the executable object file a.out, which is compiled and linked using the command:

ee209> gcc -o a.out main.c foo.c

Files main.c and foo.c consist of the following code. What is the output of a.out?

```
/* main.c */
#include <stdio.h>

static int a = 1;
int b = 2;
int c;

int main(){
   int c = 3;

   foo();
   printf("a=%d, b=%d, c=%d\n", a,
b, c);
   return 0;
}
```

```
/* foo.c */
int a, b, c;

void foo(){
   a = 4;
   b = 5;
   c = 6;
}
```

Answer: a= 1, b= 5, c= 3

(2) (4 points) Consider the following two blocks of code contained in ***separate*** files. Will the code be successfully compiled, linked, and ran? If not, at which step does the code fail?

```
/* main.c */
int i = 0;
int main() {
    foo();
    return 0;
}
```

```
/* foo.c */
int i = 1;
void foo() {
    printf("%d", i);
}
```

Answer: it will fail to link

SID:	Name:

3. (20 points) Debugging

(1) (10 points) This function should print every alternate character starting from the second character of the input (i.e., for an input of "Hello world!", the output should be "elol!"). Note: Assume that ASCII characters are given as input

```
void q2b(void) {
    while (getchar() != EOF)
        putchar(getchar());
}
```

(i) When does it produce the wrong answer?

Full (3) points if the answer describes the following.

- When the number of characters given as input is odd.
- When EOF is given in the getchar() call inside putchar()
- When EOF is detected at the alternating iteration.

- Deduct 1 point if the answer is not generalizable (only state a few cases).
- Deduct 1 point if the answer describes wrong behavior.

(ii) Rewrite the code to fix the bug.

Full (7) points if the code in the answer outputs the alternate characters without any problem.

- Deduct 3 points if the code is not executable, but correctly describes the right way to solve the bug (pseudo-code).
- Deduct 1 point if there is a minor syntactical error (e.g. missing semicolon).

Sample solution)

```
void q2b(void) {
    int c;
    for (;;) {
            c = getchar();
            if (c == EOF)
                    return;
            c = getchar();
            if (c == EOF)
                    return;
```

```
            putchar(c);
        }
    }
```

SID:	Name:

(2) (10 points) This function should return the maximum value in a non-empty array `a` of `n` integers.

```
int q2c(int *a, int n) {
    int currmax = 0, i;
    assert(a != NULL);
    assert(n > 0);
    for (i = 0; i < n; i++)
        if (a[i] > currmax)
            currmax = a[i];
    return currmax;
}
```

(i) When does this code return the wrong value?

Full (3) points if the answer describes the following.

- When the array is composed entirely of negative integers.

- Deduct 1 point if the answer is not generalizable (only state a few cases).
- Deduct 1 point if the answer describes wrong behavior.

(ii) Modify the code to correct the bug.

Full (7) points if the code in the answer returns the maximum value of an array consisting of only negative integers without any problem.

- Deduct 3 points if the code is not executable, but correctly describes the right way to solve the bug (pseudo-code).
- Deduct 1 point if there is a minor syntactical error (e.g. missing semicolon).

Sample solution)

```
int q2c(int *a, int n) {
    int currmax, i;
    assert(a != NULL);
    assert(n > 0);
```

```
    currmax = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > currmax)
            currmax = a[i];
    return currmax;
}
```

SID:	Name:

4. (20 points) Recursion

(1) (10 points) Look-and-say Sequence:

In mathematics, the look-and-say sequence is a sequence of integers where the n-th term is generated by reading the (n-1)-th term:
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, …
To generate a member of the sequence from the previous member, read off the digits (0~9) of the previous member, counting the number of digits in groups of the same digit. For example:

- 1 is read off as "one 1" or 11.
- 11 is read off as "two 1s" or 21.
- 21 is read off as "one 2, then one 1" or 1211.
- 1211 is read off as "one 1, one 2, then two 1s" or 111221.
- 111221 is read off as "three 1s, two 2s, then one 1" or 312211.

We're going to implement look-and-say by treating each member as a string.

Implement a function lookandsay(s) that takes a member of a sequence and prints out the next member by reading off the digits of member `s`, where `s` is a string (e.g., lookandsay("11") should print out "21").

Hint: Use recursion. (You can do this with less than 10 lines of code.)

Note1: printf("%c", s[0]) will print out "1" when s= "1".
Note2: You can use printf() without including the header. Assume the header file is already included.
Note3: You can assume that an input is a valid string. In other words, you don't have to handle corner cases. (e.g. string with non-integers, insufficient memory)

(write your code on the next page)

SID:	Name:

```
/* Reads term s (e.g., if s is “1” print 11 to stdio) */
void lookandsay(const char *s)
{

    char c;
    int cnt = 1;
    if (!s) return;
    if (*s==0) return;
    c = s[0];

    while (c==*(++s)) {
        cnt++;
    }
    printf("%d%c", cnt, c);
    lookandsay(s);




}
```

[Criteria]

- Deduct 0.5 points for each grammatical nit (Syntax)
- Deduct 0.5 points if argument validation statement(i.e. !s or *s==0) does not exists
- Deduct 0.5 points if printf format/order has mistake(i.e %c%d, %d%d), but has correct arguments.
- Deduct 1 points if the functions of ‘string.h’ are used. e.g) strlen()
- Deduct 3 points if there is a mistake in printing "count" variable(i.e. printf((char)cnt);)
    - No deduction: char count_start = ‘0’ & print count_start + count(integer) or count_start++;
- Deduct 4 points if there is a mistake in logic (i.e. missing single s++, wrong static usage)
- No point for incorrect logic(huge mistake)
    - E.g.) Don’t print it to stdio, use EOF or sizeof(s), wrong recursive usage

SID:	Name:

(2) (10 points) Remove duplicate in a string (array of char):
You are asked to write a recursive function that removes consecutive duplicates in a string. For example, if the input string is “aab”, the output string should be “ab”. The function modifies the array of char given as the parameter to write the resulting output string.

When you execute the main() function below, it outputs “[10, Helo World]”.

Fill in the blanks below.

Note: The C library function void *memmove(void *str1, const void *str2, size_t n) copies n characters from str2 to str1, but for overlapping memory blocks, memmove() is a safer approach than memcpy().

```
/* removeDuplicate(int len, char s[]) removes consecutive duplicates
and returns the length of resulting string. It stores the output string
in s[]. So s[] is modified */

int removeDuplicate(int len, char s[])
{
    int k;
    if (__len<=1__) return len; //2 points


    if (s[0] == s[1]) {
        k= removeDuplicate (___len-1___, ____s+1___); //2 points
        memmove(s, s+1, k+1);
        return ___k___; //2 points
    }
    k = removeDuplicate (___len-1___, ___s+1___); //2 points

    return ___k+1___; //2 points
}

int main()
{
    str[] = “Hello World”;
    int k = removeDuplicate(strlen(str), str);
    printf(“[%d, %s]\n”, k, str);
 }
```

SID:	Name:

5. (15 points) More Debugging (memory management)!

Consider the following (very buggy) program to read in a series of ints (length of 10) from stdin and report their sum on stdout:

```
1     #include <stdio.h>
2     #include <stdlib.h>
3
4     enum {LENGTH=10};
5
6     static void printSum(const int aiNums[LENGTH], size_t ulLen) {
7         int iSum = 0;
8         int *piIndex = malloc(sizeof(*piIndex));
9
10       while(*piIndex < ulLen)
11          iSum += aiNums[ulLen--];
12       printf(iSum);
13       free(aiNums);
14    }
15
16    int main(void) {
17       int iScanfReturn;
18       int *piSomeInts, *piThisInt;
19
20       piSomeInts = calloc(LENGTH, sizeof(4));
21       piThisInt = piSomeInts;
22       while((iScanfReturn = scanf("%d", piThisInt)) == 1)
23          piThisInt++;
24       printSum(piSomeInts, piThisInt-piSomeInts);
25       free(piSomeInts);
26       return 0;
27    }
```

(1) (5 points) gcc will show a warning on at least two lines. List those line numbers.

Answer: 10, 12, 13

10 (comparison of integers of different signedness),

12 (pointer from integer without a cast, i.e., missing format string)

13 (passing argument discards 'const' qualifier, i.e., free takes a pointer not a pointer to constant)

[Criteria]

Full points if all the answers are correct (5pts)

- If only 1 line is correct (3pts)

- If write more than 2 lines and there have incorrect lines, deduct 1pt per incorrect line

SID:	Name:

(2) (5 points) After fixing the lines mentioned in 5(1) (you don't have to do so!), there remain two logic bugs dealing with loop control in printSum. Briefly describe them.
Answer:
(1) *piIndex is not initialized before being used in the loop sustaining condition on line 10.
(2) ulLen-- does not evaluate to the decremented value, so the array index is initially out of bounds

Full points if all the answers are correct (5pts)
- If only one is correct (3pts)
- If write comparison of integers of different signedness in line 10 (1pt)
- If answer is correct but did not correct the ulLen-- to --ulLen, deduct (1pt)

(3) (5 points) After fixing the lines mentioned in 5(1), there remain at least two dynamic memory management issues in printSum. One is that there is no check that malloc did not return NULL. Describe another error.
Answer: Line 13 frees aiNums, not piIndex. This means that (1) piIndex is never freed, so this is a memory leak, (2) when line 25 frees piSomeInts, this is a double free.

Full points if the answer is correct (5pts)
- If write a problem about calloc return, it is not in the printSum function (0pt)

SID:	Name:

6. (25 points) Linked List and Hash Table

Below is an implementation of a data structure that stores a (key, value) pair. The key is a double type and the value is an integer type. The key ranges from [0, 1.0). Table.first is the head of the linked list and Table.array points to "short cuts". Note all nodes are connected to the linked list.

"table.c"

```
#include "table.h"

enum {BUCKET_COUNT = 100};
struct Node {
 double key;
 int value;
 struct Node *next;
};
struct Table {
 struct Node *array[BUCKET_COUNT];
 struct Node *first;
};
unsigned int hash(const double x) {
  assert(x<1.0);
  assert(x>=0.0);
  return x*100;
}
struct Table *Table_create(void) {
  struct Table *t;
  t = (struct Table*)calloc(1, sizeof(struct Table));
  return t;
}

void Table_add(struct Table *t, const double key, int value)
{
  assert(key<1.0);
  assert(key>=0.0);

  int h = hash(key);
  struct Node *p = (struct Node*)malloc(sizeof(struct Node));
  struct Node *q = t->first;
```

SID:	Name:

```
  p->key = key;
  p->value = value;
  p->next = NULL;

  if (q==NULL || q->key> key) {
    t->first = p;
    p->next = q;
    if ((t->array[h]==NULL) || (t->array[h] && t->array[h]->key > key))
      t->array[h] = p;
    return;
  }
  while (q->next!=NULL && q->next->key < key) {
    q = q->next;
  }
  // insert p after q
  p->next = q->next;
  q->next = p;
  if ((t->array[h]==NULL) || (t->array[h] && t->array[h]->key > key))
    t->array[h] = p;
}

void printTable(struct Table *t)
{
  struct Node *q = t->first;
  while (q!=NULL) {
     printf("(%lf, %d) ", q->key, q->value);
     q= q->next;
  }
  printf("\n------------\n");
  for (int i=0;i<BUCKET_COUNT;i++){
    if (t->array[i])
      printf("t->array[%d] = (%lf, %d)\n", i, t->array[i]->key, t-
>array[i]->value);
  }
}
```

SID:	Name:

(1) (10 points) Complete the search function below. You will get full credit only if you use the "shortcut" (i.e., t->array and hash()).

```
/* Table_search searches the table for the given key. On success, it
returns  0 and the value is set to the corresponding value. Otherwise, it
returns -1. */

int Table_search(Table* t, const double key, int * value)
{
  struct Node *p;
  int h = hash(key);

  if (t->array[h] == NULL)
    return -1;

  for (p=t->array[h]; p!=NULL; p=p->next)
  {
    // optional
    if (h!=hash(p->key))
      return -1;

    if (p->key == key)
    {
      *value = p->value;
      return 0;
    }
  }
  return -1;
}
```

[Criteria]

- Deduct 0.5 ~ 1 points for each grammatical nit (Syntax)
- Deduct 0.5 points if argument validation statement(i.e. t->array[h]==NULL) does not exists
- Deduct 3 points if it put value to value instead *value
- Deduct 3 points if there exists memory leakage
- Deduct 4 points if there is a mistake in logic (i.e. error in for loop)
- Deduct 7 points if it does not using "shortcut"
- No point for incorrect (huge mistake)

SID:	Name:

(2) (10 points) Table_add() provided in "table.c" is suboptimal. How can you improve it?

(i) Please write down the algorithm in steps. To get full credit, the algorithm has to cover all cases. (Assume that the keys are unique, i.e., the client only uses unique keys.) For each case, write down how the value of t->array[h] and t->first need to change.

[Solution]

```
h = hash(key)
If h == 0 do
   If t->array[h] == NULL OR t->array[h]->key > key do
      t->array[h] = p
      Insert p at the beginning of a linked list
      t->first = p
   Else do
      Traverse a linked list from the beginning and insert p
Else do
   If t->array[h] == NULL OR t->array[h]->key > key do
      Find an h-k, where t->array[h-k] is not NULL
      Traverse a linked list from t->array[h-k] and insert p
      t->array[h] = p
   Else do
      Traverse a linked list from t->array[h] and insert p
```

[Criteria]

- Idea of traversing from a middle of a linked list by using hash(). (+1 pts)
- Algorithm can handle the following cases correctly.

(a) t->array[h] is NULL (+1 pts)
(b) t->array[h]->key > key (+2 pts)
(c) t->array[h]->key < key (+2 pts)
(d) h == 0 (+1 pts)

You can get points only if you connect a new node to both previous and next nodes.

(ii) Why is your algorithm faster than the original Table_add()? Please explain it in detail.

It doesn't need to traverse all nodes by using shortcut.

(3) (5 points) To improve consistency, what would you do to the code provided ("table.c") in problem 6? (We are expecting one thing.)

printTable -> Table_print()