Spring Semester 2016

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 (1:00 PM – 3:20 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 25
Question 2	/ 15
Question 3	/ 20
Question 4	/ 15
Question 5	/ 15
Question 6	/ 10
Question 7	/ 15

Total / 115

Name: Student ID:

1. (25 points) Small Programs
   - Assume that we have included proper header files (e.g., <stdio.h>).
   - Assume that we are using 64-bit OS.

(a) (5 points) What's the output of this code snippet?

```
unsigned int i = 0xFF000000;
char x = (char) i;

printf("x = %d\n", (int)x);
```

⇨ x = 0

(b) (5 points) What's the output of this code snippet?

```
int x = 1;

if (x = 0) {
   printf("x is 0, x=%d\n", x);
} else {
    printf("x is not 0, x=%d\n", x);
}
```

⇨ x is not 0, x=0

(c) (5 points) What's the output of this code snippet?
(%zu takes an unsigned long integer)

```
char *x = "Hello World\n";
char y[] = "Hello World\n";
char **z = &x;

printf("1:%zu 2:%zu 3:%zu 4:%zu 5:%zu\n",
          sizeof(x), sizeof(y), sizeof(z),
          sizeof(*x), sizeof(*z));
```

⇨ 1:8 2:13 3:8 4:1 5:8

Name: Student ID:

(d) (5 points) What's the output of this code snippet?

```
void f(int x)
{
    if (x /10) f (x/10);
    if (x % 10) putchar ('0' + x % 10);

}

…
f(30405);
```

⇨ 345

(e) (5 points) What's the output of this code snippet?

```
int i, j;
i = 100;
j = 200;
i = i ^j;
j = i ^j;
i = i ^j;
printf("i=%d, j=%d\n", i, j);
```

⇨ i=200, j=100

Name: Student ID:

2. (15 points) A C runtime library function, strstr(), finds a substring in a string. That is,

```
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).

Unfortunately, strstr() is *case-sensitive*, so in the code snippet below,

```
char *x = "hello world\n";
char *p1, *p2;
p1 = strstr(x, "WoRLd");
```

p1 will be NULL, since x does not contain "WoRLd".

Here, we want to write a ***case-insensitive*** version of strstr() called strcasestr() whose function prototype is

```
char * strcasestr(const char* haystack, const char* needle);
```

and

```
p2 = strcasestr(x, "WoRLd");
```

p2 will point to the address of 'w' in x.

Please fill in the function body in the next page. For brevity, we have renamed the parameter names to s (haystack) and n (needle). That is, you need to find n in s without considering its case. Note that you **can't** call any C runtime library functions except toupper() or tolower(), but you can define and call your ***own*** functions outside strcasestr(). For brevity, you can assume both s and n are non-null, that is, you don't need to check if they are NULL. Your code should compile without any errors or warning messages with gcc209.

You can use these C runtime library functions. No need to include a header file.

```
int toupper(int c);
int tolower(int c);
```

Name: Student ID:

```
int match(const char* s1, const char* s2)
{
    while (*s2) {
        if (*s1 == '\0')
            return 0;
        if (toupper((int)*s1) != toupper((int)*s2))
            return 0;
        s1++; s2++;
    }
    return 1;
}

char *strcasestr(const char *s, const char *n)
{
    if (*n == '\0') return NULL;
    while (*s) {
        if (match(s, n)) return (char *)s;
        s++;
    }
    return NULL;
}
```

Name: Student ID:

3. (20 points) Number System

(a) (5 points) short i = (short)-123;

Represent i in the binary format (assume short int is 16 bits)

⇨ 11111111 10000101

(b) (5 points) What's the output of this code snippet?

```
short i = 32768;
printf("i = %hd\n", i-1);
```

⇨ i = 32767

(c) (5 points) What's the output of this code snippet?

```
printf("val = %hu\n", (unsigned short)-1);
```

⇨ val = 65535

(d) (5 points) What's the output of this code snippet? (Assume we are using 64-bit OS.)

```
char *p[] = {"EE 209", "Midterm Exam"};
printf("val = %c sizeof(p)=%zu\n", *(*(p + 1)+3), sizeof(p));
```

⇨ val = t sizeof(p)=16

Name: Student ID:

4. (15 points) Swap Functions

(a) (5 points) Swap two integers

```
void swap_int(int *i, int *j)
{
   int temp;
   temp = *i;
   *i = *j;
   *j = temp;
}

int a =10, b = 20;
swap_int(&a, &b);
printf("a = %d, b = %d\n", a, b);
```

I want to see the output as "a = 20, b = 10\n". Please fill out the above swap function. Note that you should swap the two values regardless of the actual values of a and b instead of doing any trivial assignment (e.g., *i = 10, *j= 20;) Be efficient. Avoid using any unnecessary variables and assignments.

Name: Student ID:

(b) (10 points) Swap two integer pointers

```
void swap_pointers(int **ppa, int**ppb)
{
    int * temp;
    temp = *ppa;
    *ppa = *ppb;
    *ppb = temp;
}
int a = 10, b = 20;
int *pa = &a, *pb = &b;
swap_pointers(&pa, &pb);
printf("a = %d, b = %d\n", *pa, *pb);
```

I want to see the output as "a = 20, b = 10\n". Please fill out swap_pointers() above. Note that you should swap the two pointers regardless of their actual values. Be efficient. Avoid using any unnecessary variables and assignments.

Name: Student ID:

5. (15 points) Dynamic Memory Allocation and Debugging

```
#define MYLIE1 “I Like Midterm Exam”
#define MYLIE2 “I Like Midterm Exam Since It is Easy”
void midterm_function(void)
{   /* the line number on the left is invisible to compiler */
    1. char *p = MYLIE2;
    2. strcpy(p, MYLIE1);
    3. p = malloc(sizeof(MYLIE1));
    4. if (p == NULL) return;
    5. strcpy(p, MYLIE1);
    6. p = realloc(p, strlen(MYLIE2));
    7. if (p == NULL) return;
    8. strcpy(p, MYLIE2);
    9. free(p);
}
int main(void)
{
   midterm_function();
   return 0;
}
```

(a) (5 points) I compiled this code, and run it. But the program crashes. At which line does the program crash? Please explain why the program crashes.

⇨ The program crashes at line 2 since the program attempts to write to a read-only memory section (string literal)

(b) (5 points) I commented out the line (only that line) at which the program crashes. I compiled the code and run it again, and it crashes sometimes (not deterministic) even if I have lots of available memory. At which line does the program crash? Please explain why the program crashes.

⇨ The program crashes at line 8, since p is too small to hold MYLIE2. It should have allocated strlen(MYLIE2) + 1 bytes.

(c) (5 points) So, I fixed the line in (b), and the program now runs well. I was so happy and I showed my code to my friend. He pointed out one problem in the code. What could that be? Please rewrite the code to fix the problem.

⇨ realloc() might return NULL, and it would lead to a memory leak. To fix the code,

```
char *q;
.,..
q = realloc(p, strlen(MYLIE2) + 1);
if (q == NULL) {
    free(p);
    return;
}
p = q;
…
```

Name: Student ID:

7. (10 points) Stack Abstract Data Type

We are writing a stack module that pushes and pops a string. For this question, we implement only Stack_PushString() declared in stack.h. Stack_PushString() pushes an input string onto a stack. We use a linked list to implement it. That is, internally, it allocates an ItemT object, copies the input string (to own it), and inserts the object into the head node to place it at top of the stack. It returns 1 when it successfully pushes a string onto a stack or 0 if there is any error. In case of an error, you are responsible for releasing all memory that you have allocated. You can use any C runtime library functions. Write the function body in the next page.

(stack.h)

```
struct stack;
typedef struct stack* StackT;
StackT Stack_New(void);
int Stack_PushString(StackT t, const char* s);
…
```

(stack.c)

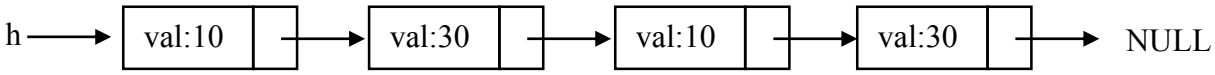
```
typdef struct item {
    char *p;
    struct item* next;
} ItemT;
struct stack {
    ItemT *head;
};
StackT Stack_New(void)
{
    StackT t;
    t = calloc(1, sizeof(StackT));
    if (t == NULL) return NULL;
    return t;
}
```

```
int Stack_PushString(StackT t, const char *s)
{
   ItemT *item;

   assert(t != NULL && s != NULL);
   if (t == NULL || s == NULL)
      return 0;

   item = malloc(sizeof(ItemT));
   assert(item != NULL);                              // optional
   if (item == NULL)
      return 0;

   item->p = strdup(s);
   assert(item->p != NULL);                           // optional
   if (item->p == NULL) {
       free(item);
       return 0;
   }

   item->next = t->head;
   t->head = item;
   return 1;
}
```

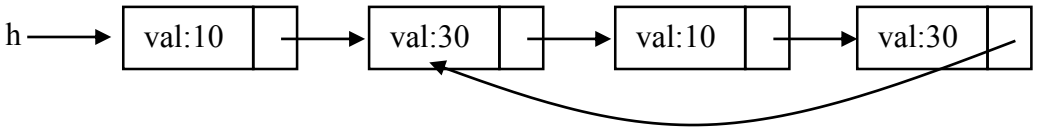
Name: Student ID:

6. Extra Credit (15 points) Cycle in the List

A linked list is said to have a cycle if the next node points back to some previous node in the list. A cycle in a linked list is undesirable since your code falls into an infinite loop while traversing the node in the list. The following code reveals whether there is a cycle in a list whose head pointer is h. That is, has_cycle() returns 1 if there is a cycle in the list or 0 if there is no cycle. Fill in the body of the function below such that the code itself does not fall into a cycle even if there is one. (Hint: use two pointers and make them traverse the nodes)



Example 1: A linked list without a cycle



Example 2: A linked list with a cycle

```
typedef struct list {
    int val;                /* don't need to be unique among the nodes in the list */
    struct list *next;      /* next node pointer */
} List;
```

Name: Student ID:

```
int has_cycle(List *h)
{
    List *p, *q;

    p = h;
    if (p == NULL) return 0;
    q = h->next;
    if (q == NULL) return 0;
    while (1) {
        if (p == NULL || q == NULL) return 0; /* reached the end, so no cycle */
        if (p == q) return 1;            /* p and q met, so it's a cycle */
        p = p->next;                     /* p advances (by one node) to the next node */
        q = q->next;
        if (q == NULL) return 0;    /* q reached the end, so no cycle */
        q = q->next;                /* q advances by two nodes */
     }
    return 1; // unreachable
}
```

Algorithm description

p starts at the first node while q starts at the second node. Whenever p or q is NULL, there's no cycle. Whenever p and q are equal, there is a cycle. Each time, p advances to the next node (one hop) while q advances forward by two nodes (next node's next node).