Fall Semester 2020

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 through chat, 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 165 minutes (9:00 AM – 11:45 AM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 15
Question 2	/ 20
Question 3	/ 13
Question 4	/ 20
Question 5	/ 32
Total	/ 100

Name: Student ID:

1. (15 points) Numbers and Bit Operations

(a) **(5 points)** Consider 6-b it signed (two's complement) integer. Fill in the below table, except for gray boxes. **Tmin** indicates the minimum value of the signed integer, where **Tmax** is the maximum value of the signed integer. Put N/A if the answer cannot be expressed in 6-bit signed binary.

Expressions	Decimal	6-bit binary (signed)
	-7	111001
	21	010101
Tmin	-32	100000
Tmax + 1	32	N/A or 100000
-Tmax	-31	100001
–Tmin	32	N/A or 100000

(b) **(4 points)** Overflow occurs when # of bits is insufficient for certain computation. Answer the followings:

Part 1. Consider 4-bit addition (both operands and result are all **4-bit integer**). Provide an example where **overflow occurs for signed integer, but does not occur for unsigned integer**.

0111 + 0001 = 1000

Part 2. Again, consider 4-bit addition (both operands and result are all 4-bits). Provide an example where overflow occurs for **both signed and unsigned integers.**

1111 + 1000 = 0111

Overflow occurs for unsigned when upon carry. Overflow occurs for signed when MSB changes. There can be multiple answers.

Name: Student ID:

(c) **(6 points)** For the following, assume x is a signed (two's complement) 32-bit integer. **Tmin** is the minimum value of signed 32 bit integer.

- Function body can only have a **single expression/statement (i.e., return)**
- Only a set of bit operator are allowed: {+, &, |, !, ~, ^, <<, >>}
- Use one bit operator or combination of multiple bit operators

Example. `int minusOne(void)` that returns a value of -1

```
int minusOne(void)
  { return ~0;
}
```

Part 1. Implement `int tMin(void)` that returns the bit sequence corresponding to 32-bit Tmin (i.e., the minimum value of signed 32 bit integer)

Ans:

```
int tMin(){
   return 1<<31;
}
```

Part 2. Implement `int isPositive(int x)` that returns 1 if x is non-negative or 0 otherwise

Ans:

```
int isPositive(int x) {
   return !(x>>31)
}
```

Name: Student ID:

2. (20 points) Pointers and Arrays

(a) **(4 points)** What's the output of this code snippet?

```
#include <stdio.h>

void main() {
  int num_array1[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int *ptr1, *ptr2;

  ptr1 = num_array1+2;
  ptr2 = &num_array1[5];

  printf("num_array1: %p\n", num_array1);
  printf("ptr2: %p\n", ptr2);
  printf("ptr2-ptr1: %d\n", (int)(ptr2-ptr1));
}
```

Output:

num_array1: 0x7fffd068d580

ptr2: 0x7fffd068d594

ptr2-ptr1: 3

Name: Student ID:

(b) **(10 points)** In class, we learned the array name can be used as a pointer. Similarly, in a 2-dimiensional array, the array name points to the beginning of the entire 2-dimensional array. Also, array name with only row (without column) points to the beginning of that row (look carefully into the output of the first `printf` function). Knowing this, what is the output of this code snippet?

```
#include <stdio.h>

void main() {
  int num_array2[4][5] = {{0, 1}, {2, 3, 4}, {5, 6, 7, 8}, {9}};
  int *ptr3;

  printf("Hint: num_array2[1] (%p), &num_array2[1][0] (%p),
  num_array2+1 (%p) are all same!\n", num_array2[1],
  &num_array2[1][0], num_array2+1);

  ptr3 = *(num_array2 + 2);

  printf("*ptr3: %d\n", *ptr3);
  printf("*(&num_array2[1][2]+1):%d\n", *(&num_array2[1][2]+1));
  printf("*(num_array2[2] + 5): %d\n", *(num_array2[2] + 5));
  printf("sizeof(num_array2) %d\n", (int)sizeof(num_array2));
  printf("sizeof(num_array2[3]) %d\n", (int)sizeof(num_array2[3]));
}
```

Output:

Hint: num_array2[1] (0x7ffd96f1d454), &num_array2[1][0] (0x7ffd96f1d454), num_array2+1 (0x7ffd96f1d454) are all same!

*ptr3: 5

*(&num_array2[1][2]+1): 0

*(num_array2[2] + 5): 9

sizeof(num_array2): 80

sizeof(num_array2[3]): 20

Name: Student ID:

(c) **(6 points)** What is the output of this code snippet?

```
#include <stdio.h>

void main() {

  char *class[]={"KAIST", "EE209", "Spring", "2020"};

  printf("**class+3: %c\n", **class+3);
  printf("*(*class+3): %c\n", *(*class+3));
  printf("**(class+3): %c\n", **(class+3));
}
```

Output:

**class+3: N

*(*class+3): S

**(class+3): 2

Name: Student ID:

3. (13 points) Recursive functions

Write a **recursive** function that prints binary of a positive integer, by filling in the white box.

```
#include <stdio.h>

void print_binary(unsigned int n);

int main() {
  unsigned int number;

  printf("Input a positive integer:\n");
  scanf("%u", &number);
  printf("In binary: ");
  print_binary(number);
  putchar('\n');

  return 0;
}

void print_binary(unsigned int n) {
```

Fill in here

```
}
```

Ans:

```
void print_binary(unsigned int n) {
   int r;
   r = n%2;
   if(n >= 2)
      print_binary(n/2);
   putchar(r==0 ? '0':'1');
   return;
}
```

Name: Student ID:

4. (20 points) Linked list and hash table

(a) **(14 points)** Below is the Node and Table structure we learned in class to build hash table.

```
enum {BUCKET_COUNT = 1024};

struct Node{
  const char *key;
  int value;
  struct Node *next;
};

struct Table {
   struct Node *array[BUCKET_COUNT];
};
```

Write a `Table_delete` function that removes node(s) with the corresponding key. To avoid key values overwritten by the user (which makes the hash table malfunction) our data structure owns a copy of the key. This is done by the below code, as we discussed in the class.

```
void Table_add(struct Table *t, const char *key, int value)
{
   …
   struct Node *p = (struct Node*)malloc(sizeof(struct Node));
   p->key =(const char*)malloc(strlen(key) + 1);
   strcpy(p->key, key);
   …
}
```

Now, implement `Table_delete` function when the data structure owns a copy of the key. ***Hint:*** As we've learned in modularity, well-designed module manages resource consistently (i.e., a module should free a resource if and only if the module has allocated that resource). The function should also satisfy the below requirements:

- Return 1 if the node is successfully removed
- Return 0 if the node with the corresponding key is not found
- After deletion, all other nodes should remain accessible and should be in the same order.
- Assume duplicated keys, where they are all deleted.
- You may use library functions (e.g., `strcmp`)
- Use `hash` function. But you may NOT call other hash table functions (e.g., `Table_search`).

Name: Student ID:

Ans:

```
int Table_delete(struct Table *t, const char *key) {
  struct Node *p, *q;
  int matched = 0;
  int h = hash(key);

  // Loop to delete the first node with the matching key
  while (t->array[h] && strcmp(t->array[h]->key, key) == 0) {
    q = t->array[h];
    t->array[h] = t->array[h]->next;
    free(q->key);
    free(q);
    matched = 1;
  }

  // Search to delete non-first nodes with the matching key
  for (p = t->array[h]; p != NULL;) {
    if (p->next && strcmp(p->next->key, key) == 0) {
      q = p->next;
      p->next = q->next;
      free(q->key);
      free(q);
      matched = 1;
    }
    else
      p = p->next;
  }
  return matched;
}
```

Name: Student ID:

(b) **(6 points)** Below is a silly and meaningless linked list. What's the output of this code? Drawing the list should be helpful to track down the linkage and find the answer.

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

typedef struct t{
    struct t *next;
    int val;
} rec, *rec_ptr;

int main() {
    rec mystruct1 = {NULL,52};
    rec mystruct2 = {NULL,21};
    rec_ptr t1, t2, t3;
    t1       = (rec_ptr) malloc(sizeof(rec));
    t1->val  = 67;
    t1->next = (rec_ptr)malloc(sizeof(rec));

    t1->next->val  = 90;
    t1->next->next = t1;

    t1       = t1->next;
    t2       = &mystruct1;
    t2->next = t1;
    t3       = &mystruct2;
    t3->next = t1->next;
    t2->next = t3;

    printf("%d %d %d %d\n", t1->next->next->val,
       t3->next->val, mystruct1.next->val,
       mystruct1.next->next->val
    );
}
```

Ans: 90 67 21 67

Name: Student ID:

5. (32 points) String Manipulation

(a) **(12 points)** We have implemented different string manipulation functions in assignment 2, Similarly, we implement `StrPtrBreak` function following the below man page and example usage.

Man page:

```
NAME
    StrPtrBreak - search a string for any of a set of bytes

SYNOPSIS
    char * StrPtrBreak (const char s1[], const char s2[]);

DESCRIPTION
    The StrPtrBreak() function locates the first occurrence in
    the string s1 that matches any character in s2.

RETURN VALUE
    StrPtrBreak() returns the pointer to the character in s1 that
    first matches to any character in s2. Return NULL if no such
    character is found.
```

Usage example:

```
const char s1[] = "Apple";
const char s2[] = "Orange";
char *ret;
ret = StrPtrBreak(s1, s2);
```

The above example code should return a pointer with an address to ‘A’. Please implement `StrPtrBreak` under the below requirements:

- Use array notation (instead of pointer notation), as in your assignment 2.
- Do not use library functions (e.g., `strstr`)
- Case insensitive
- Assume we are using ASCII

Name: Student ID:

```
const char * StrPtrBreak(const char s1[], const char s2[]) {
  int s1Idx = 0, s2Idx = 0;
  char c1 = 0, c2 = 0;

  while((c1 = s1[s1Idx++]) != 0) {
    for(s2Idx = 0; (c2 = s2[s2Idx++]) != 0;) {
      if(c1 == c2)
        return (&s1[s1Idx-1]);
      else if(c1 >= 'A' && c1 <='z' && c2 >= 'A' && c2 <= 'z') {
        if(c1-c2 == 'a'-'A' || c2-c1 == 'a'-'A')
          return (&s1[s1Idx-1]);
      }
    }
  }
  return NULL;
}
```

Note: const in the return type removes the warning (no deduction for missing it)

Name: Student ID:

(b) **(20 points)** Suppose a code `GetKAISTEmail.c` takes input from `stdin` and outputs KAIST email ids. Consider `text_with_emails.txt` with below content:

```
I hope you are doing well in the exam!
m.jung@kaist.ac.kr and songmin@kaist.ac.kr are instructors.
EE209A's head TA email is mkwon@camelab.org.
EE209B's head TA email is jwj8615@kaist.ac.kr.
```

Inputting `text_with_emails.txt` into `GetKAISTEmail` yields the below result:

```
$gcc209  -o GetKAISTEmail GetKAISTEmail.c
$./GetKAISTEmail < text_with_emails.txt
List of KAIST email IDs:
m.jung
songmin
jwj8615
```

Implement GetKAISTEmail.c by filling in the box in the below code.

- Use only `strstr`, `strcpy`, `printf` functions
- Use only the variables declared in the problem. Do not declare additional variables.
- Assume each line (including '\n') is no more than 1023 bytes.
- Assume emails are not written across two lines
- ***Hint*: email IDs cannot include a space**.

```
/* GetKAISTEmail.c */
#include <stdio.h>
#include <string.h>
#define MAX_LINE 1024

void main() {
  char line[MAX_LINE];  // Store a line of text
  char email[MAX_LINE]; // Store email id
  char *pStart, *pEnd;  // Access memory with email IDs

  printf("List of KAIST email IDs:\n");

  while(fgets(line, sizeof(line), stdin)){
    Fill in here
  }
}
```

Name: Student ID:

```
#include <stdio.h>
#include <string.h>
#define Max_LINE 1023

int main() {

   char line[Max_LINE],email[Max_LINE];
   char *pStart, *pEnd;

   printf("List of KAIST email IDs:\n");

   while(fgets(line, sizeof(line), stdin)){
      while((pEnd = strstr(line,"@kaist.ac.kr"))!=NULL) {
         pStart = pEnd-1;
         while((*(pStart-1)) != ' ' && (*(pStart-1)) != '\t' && pStart != line)
            pStart--;
         *pEnd = '\0';
         strcpy(email, pStart);
         printf("%s\n", email);
         strcpy(line, ++pEnd);
      }
   }
}
```