Fall Semester 2015

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 then not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. You have 140 minutes (9:00 AM – 11:20 AM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 10
Question 2	/ 15
Question 3	/ 10
Question 4	/ 10
Question 5	/ 10
Question 6	/ 15
Question 7	/ 15
Question 8	/ 15
Question 9	/ 10
Total	/ 110

Name: Student ID:

1. (10 point) For the arithmetic operation 75 - 91, translate the decimal numbers to 2's complement form, complete the arithmetic operation, and translate the result back into decimal. Assume an 8-bit word size.

Answer)

```
  75 (base 10) = 01001011 (base 2)
  91  base 10) = 01011011 (base 2)
-91 (base 10) = 10100101 (base 2)

  +75 (base 10) = 01001011 (base 2)
+-91 (base 10) = 10100101 (base 2)
---------------------------------------------
    -16 (base 10) = 11110000 (base 2)
```

Name: Student ID:

2. (15 point) Translate each of the following English-language descriptions into a C declaration.

(a) (5 point) Structure person has three components: name (a pointer to a structure that contains two components fname (a character string), and lname (a character string)); dob (an array of 3 integers), and parent (a pointer to a person).

Answer)

```
struct person
      {
          struct {char *fname, char *lname} *name;
          int dob[3];
          struct person *parent;
};
```

typedef을 사용하거나, 내부 structure를 다른 structure로 선언하고, struct person을 만든 경우도 맞음.

(b) (5 point) Declare variable **`employees`** as an array of 10 pointers to structure person.

Answer) struct person *employees[10];

Name: Student ID:

(c) (5 point) Suppose variable **`employees`** is a pointer to a person structure, and variable **`numemps`** is an integer. Show the code to dynamically setup employees (i.e., dynamic memory allocation) as an array of size numemps, with everything allocated initialized to zero.

Answer)

```
struct person *employees;
	int numemps;
	...
	employees = (struct person*)calloc(numemps, sizeof(struct person));
```

zero로 initialize하라고 햇으므로, calloc을 사용해야 함. 하지만, malloc을 사용하고, 뒤에 for loop를 통해서 초기화한 코드도 맞음.

Name: Student ID:

3. (10 point) What is the output of the following program?

```
#include <stdio.h>

int main(void) {
    char *array[3][3]=
           {  {":|(", "Hope", "you"},
              {"have", "a", "good"},
              {"fall", "break",":|}"}
           };
    char *(*p)[3]=array;
    char **q=*(array+1);

    printf("1 : %s\n",array[0][2]);
    printf("2 : %s\n",*(array[1]));
    printf("3 : %s\n",*(*(array+1)+1));
    printf("4 : %s\n",*(q+2));
    printf("5 : %s\n",*(*(p+2)+1));

    return 0;
}
```

Answer)

1 : you 2 : have 3:a 4 : good 5 : break

Explanation:
array[0][2]
= "you"
*(array[1])
= *(array[1] + 0)
= array[1][0]
= "have"
((array + 1) + 1)
= *(array[1] + 1)
= array[1][1]
= "a"
*(q + 2)
= *(*(array + 1) + 2)
= *(array[1] + 2)
= array[1][2]
= "good"
((p + 2) + 1)
= *(*(array + 2) + 1)
= *(array[2] + 1)
= array[2][1]
= "break"

Name: Student ID:

4. (10 point) What string does the function **f()** below return? Explain your answer.

```
char* f(unsigned int n){
  int i, numbits = sizeof(unsigned int) * 8;

  char* ret = (char *) malloc(numbits + 1);

  for (i=numbits-1; i>=0; i--, n>>=1)
    ret[i] = '0' + (n & 1);

  ret[numbits] = '\0';

  return ret;
}
```

Answer)

The code produces a string of the unsigned integer n's binary representation.
The variable numbits is the number of bits needed to represent an unsigned int (i.e., the number of bytes multiplied by 8 bits/byte). The malloc() allocates enough space to store one character for each bit, plus room for the '\0' to terminate the string. The for loop starts at the position for the last character and proceeds up to and including the 1st character, assigning the ith character to the ith bit in the unsigned int. The assignment to ret[i] extracts the last bit of the current value of n (which is shifting out the rightmost bit on each iteration of the for loop), and assigns a '0' if the value is 0, and a '1' if the value is 1. Then, the string is terminated with '\0' and the function returns the pointer to the string.
A common mistake was to assume the code reversed the order of the bits, because the loop counts backwards. The "counting backwards" is necessary because the "n & 1" extracts the last bit, rather than the first, and this bit should appear at the end of the string.

Name: Student ID:

5. (10 point) What does the program output when executed?

```
int main()
{
   int a = 14;
   int b = 5;
   int c = 29;
   unsigned short x = 0;
   x = a << 12;
   x |= (b & 0x007) << 8;
   x |= c & 0xff;

   printf("%x\n",x);

return 0;
}
```

Answer)

e51d

Explanation:
x = a << 12
x = 14 << 12
x = 0x0000000e << 12
x = 0x0000e000
x |= (b & 0x007) << 8
x |= (5 & 0x007) << 8
x |= (0x0005 & 0x0007) << 8
x |= (0x0005) << 8
x |= 0x0500
x = x | 0x0500
x = 0xe000 | 0x0500
x = 0xe500
x |= c & 0xff
x |= 29 & 0xff
x |= 0x001d & 0x00ff
x |= 0x001d
x = x | 0x001d
x = 0xe500 | 0x001d
x = 0xe51d

Name: Student ID:

6. (15 point) For each of the following code fragments, determine whether or not it is likely to cause a run-time program error (assume that the code compiles without warnings or errors). If there is a run-time program error, please write "ERROR" next to it and describe the problem in one or two short sentences. Otherwise, simply write "OK" next to it.

a) (5 point)

```
 void *pVoid = malloc(8 * sizeof(int));
int *pInt = pVoid;
free(pInt);
```

a) OK

b) (5 point)

```
float a[] = { 1.0, 0.0, 1.0, 0.0 };
float *b = &a[2];
float *c = b-- + 1;
float result = *b / *c;
```

b) Error

```
float a[] = { 1.0, 0.0, 1.0, 0.0 };
float *b = &a[2];
```

b points to memory that contains 1.0.

```
float *c = b-- + 1;
```

The expression b-- decrements b so it points to memory that contains 0.0. But it evaluates to the prior value of b, which points to memory that contains 1.0. The expression b-- + 1 thus evaluates to the address of memory that contains 0.0. That address is assigned to c.

```
float result = *b / *c;
```

*b evaluates to 0.0. *c evaluates to 0.0. Thus the expression is identical to (0.0 / 0.0). It evaluates to NaN (not a number).

Name: Student ID:

c) (5 point)

```
 void *my_alloc(void *ptr, int size)
{
    if (ptr)
       return realloc(ptr, size);
    else
       return malloc(size);
 }

 void my_free(void *ptr)
 {
    free(ptr);
    ptr = 0;
 }

 int main()
 {
    void *ptr = my_alloc(0, 4);
    my_free(ptr);
    ptr = my_alloc(ptr, 4);
    my_free(ptr);
    return 0;
  }
```

c) Error

```
      void *my_alloc(void *ptr, int size)
      {
         if (ptr) return realloc(ptr, size);
         else return malloc(size);
      }
      void my_free(void *ptr)
      {
free(ptr);
ptr = 0; }
int main() {
         void *ptr = my_alloc(0, 4);
```

Calls malloc to allocate 4 bytes of memory, and assigns the address of that memory to ptr.

```
         my_free(ptr);
```

Frees that memory, but does not change the value of ptr. ptr still points to the freed memory.

```
         ptr = my_alloc(ptr, 4);
```

Calls realloc to change the size of the block of memory to which ptr points. But that memory has been freed. The results are unpredictable.

```
         my_free(ptr);
return 0; }
```

Name: Student ID:

7. (15 point) Consider the following function that converts an integer to a string, where the **`sprintf()`** function "prints" to a formatted string, e.g., **`sprintf(retbuf, "%d", 72)`** places the string **`"72"`** starting at the location in memory indicated by the address **`retbuf`**:

```
char *itoa(int n) {
   char retbuf[5];
   sprintf(retbuf, "%d", n);
   return retbuf;
}
```

(a) (5 point) Identify two serious bugs in this function.

Answer)

The variable retbuf is a local variable, which is stored (temporarily) on the stack. As such, the "return" statement returns a pointer to memory that is no longer allocated after the function ends. In addition, the definition of retbuf[5] does not necessarily add enough space to store the string, depending on the size of the integer.

Name: Student ID:

(b) (10 point) Rewrite the function to fix these bugs.

```
char *itoa(int n) {
    int size = 0;
    int temp = n;
    char *retbuf;
    /* Count number of decimal digits in n */
   while (temp /= 10)
        size++;
size++;
    /* If n is negative, add room for the "-" sign */
    if (n < 0)
size++;
    retbuf = (char *) malloc(size + 1);
    assert(retbuf != NULL);
    sprintf(retbuf, "%d", n);
    return retbuf;
}
```

A common mistake was to mishandle the case where n is 0, which requires 1 character
(rather than 0 characters).
Another common mistake was to assume that n is an unsigned int, and not allocated space for the minus sign.
Another common mistake was to omit the "assert(retbuf)" after the call to malloc, though no points were taken off for this.
Another mistake a few students made was to use "sizeof(n)" to compute the length; this is incorrect because it returns the number of bytes in a "int", not the number of digits in the decimal representation of n.
Some students extracted each of the decimal digits of n, using code similar to question 1b (though modified to manipulate n in base 10 rather than base 2). This is perfectly valid (and, as such, no points were taken off), though using "sprintf" is simpler.
An interesting, and clever, answer was to create a large array of characters as a local variable, use sprintf to place the string representation of n in the array, use strlen() to compute the length of the string, use malloc() to allocate the appropriate amount of space to retbuff, and then copy the string from the local variable to retbuf.

다양한 형태의 답이 나올 수 있음. 중요한 것은 rebuf 를 dynamic memory allocation 을 해서 하는가이고, 여러가지 corner case 는 기준을 만들어서, 채점을 해야 할 것임.

Name: Student ID:

8. (15 point) The program shown on the page 13 should output the following:

```
This is a fall00 midterm question.
his is a fall00 midterm question.T
is is a fall00 midterm question.Th
s is a fall00 midterm question.Thi
 is a fall00 midterm question.This
is a fall00 midterm question.This 
s a fall00 midterm question.This i
 a fall00 midterm question.This is
a fall00 midterm question.This is 
 fall00 midterm question.This is a
```

However, the program has the memory-related bug (it may not have any compile error, but it may generate segmentation fault or generates core dumps).

(a) (10 point) Identify and fix the bug.

Name: Student ID:

(b) (5 point) After fixing the bug, explain what problem might still arise if printMatrixis provided as a library function to be used in big programs. Provide a very short answer.

Answer)

(a) In the line

```
b[i] = (CHARBUF) malloc(sizeof(char) * (sizeof(a)+1));
```

the expression sizeof(a) is always 4. That is incorrect.
The line should be:

```
b[i] = (CHARBUF)malloc(sizeof(char) * (strlen(a)+1));
```

(b) Each call to printMatrix creates garbage. That is, printMatrix contains a memory leak.

Name: Student ID:

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

typedef char * CHARBUF;

/* this function prints a 10-line matrix based on the content of
* argument a. If strlen(a)<10, it will print an error message.
*/

void printMatrix(CHARBUF a)
{
   CHARBUF b[10];
   int i;
   char *to_ptr, *from_ptr;

   if (strlen(a) < 10) {
      fprintf(stderr, "string length must be larger than 10\n");
      return;
   }

   for (i=0;i<10;i++) {
      b[i] = (CHARBUF)malloc(sizeof(char) * (sizeof(a)+1));
      if (b[i]==NULL) {
         fprintf(stderr, "no more memory available.\n");
         return;
      }
      for (to_ptr = b[i], from_ptr = a+i; (*from_ptr) != 0; )
         *to_ptr++ = *from_ptr++;
      for (from_ptr = a; from_ptr != a+i; )
         *to_ptr++ = *from_ptr++;
      *to_ptr = 0;
   }

   for (i=0;i<10;i++)
      printf("%s\n",b[i]);
}

void main(void)
{
   printMatrix("This is a fall00 midterm question.");
}
```

Name: Student ID:

9. An undergraduate student wrote a function to print out characters in string in the reverse order. (e.g., given “abc” the program needs to print out “cba”.) She wrote the function in three lines of code. Fill in the blanks using two statements including **`printf()`**. For example to print a character you may use **`printf("%c", ch)`**, where **`ch`** is of type char. You cannot use any other standard library function.

   Note she did not use any loop in her program.

```
void print_reverse(const char* str)
{
      if (*str!='\0') {

      ____________________________________________;

      ____________________________________________;

     }
}
```

Answer)

```
void reverse(const char * str)
{
   if (*str!='\0')
   {
      reverse(str+1);
      printf("%c", *str);
   }
}
```