Spring Semester 2017

KAIST EE209

Programming Structures for Electrical Engineering

# Mid-term Exam

Name: _____

Student ID: _____

This exam is open book and notes. You should not _share_ any books/notes with your students during the test. 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 165 minutes (1:00 PM – 3:45 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1 _____ / 20

Question 2 _____ / 20

Question 3 _____ / 20

Question 4 _____ / 20

Question 5 _____ / 20

Total _____ / 100

Name: Student ID:

1. (20 points) Small Programs
   - Assume that we have included proper header files (e.g., <stdio.h>).
   - Assume that we are using 64-bit OS.
   - %zu prints an unsigned long integer

   (1) (5 points) What's the output of this code snippet?

```
struct node {
    long int key, value;
    struct node* next;
};
struct node a[10], *p = a, *q = p + 1;

printf("1: %zu 2: %zu 3: %zu 4: %zu 5: %zu\n",
sizeof(a), sizeof(q), sizeof(*p), sizeof(*q->next),
sizeof(25.3));
```

   ⇨ 1: 240 2: 8 3: 24 4: 24 5: 8

   (2) (5 points) Code for the problem set (2):

```
#define INIT_SIZE 1024

char* p = malloc(sizeof(char) * INIT_SIZE);
if (p == NULL) {
    fprintf(stderr, "malloc failed\n");
    exit(-1);
}
…

char *tmp = realloc(p, sizeof(char) * 2 * INIT_SIZE);
if (tmp == NULL) {
    free(p);
    fprintf(stderr, "can't go on due to lack of memory);
    exit(-1);
}
memset (p + INIT_SIZE, 0, INIT_SIZE);
…
```

   (a). Assuming we have enough dynamic memory, what kind of undesirable behavior could this code produce? (1 point)

   ⇨ It sometimes crashes due to memory access violation (SEGFAULT)

(b). What causes the undesirable behavior in (b)-1? (1 point)

⇨ realloc() may allocate the new memory at a different address than p while it frees the existing memory, but memset() still accesses the previously allocated memory area that could have been deallocated.

(c) Fix the code such that it avoids the problem. (3 points)

⇨ Replace

memset(p + INIT_SIZE, 0, INIT_SIZE);

with

p = tmp;
memset(p + INIT_SIZE, 0, INIT_SIZE);

(3) (5 points) Code for the problem set (3):

```
#include <string.h>

char *p = "Hello 0123\056789";

printf("1:%zu 2:%zu\n", strlen(p), sizeof(p));
p[9] = 'x';
printf("3:%zu 4:%zu\n", strlen(p), sizeof(p));
```

(a). What's the output of the first printf()? (3 points)

⇨ 1: 14 2: 8 (\056 is recognized as an octal character.)

(b). Will you see the output of the second printf()? If so, what's the output? If not, what happens? (2 points)

⇨ No. The program crashes with memory access violation. p[9] is part of read-only memory that should not be written to. (Note that a string constant is allocated at read-only memory section)

Name: Student ID:

(4) (5 points) What's the output of this code snippet?

```
void f(int x)
{
  printf("%d", x);
  if (x < 9) f(x+1);
  printf("%d", x)

}

…
f(1);
fflush(stdout);
```

⇨ 123456789987654321

Name: Student ID:

## 2. (20 points) String Manipulation

### (1) (10 points) String to Integer conversion

`atoi() (or atol(), strtol(), strtoll(), etc.)` converts a C string into an integer. For example, `int n = atoi("72");` then `n` becomes 72. Your job is to write a function, `int StrToInt(const char *s)` that converts `s` into an integer and returns it. Assume that the string can be converted into a signed integer without an error. That is, you do not need to handle error cases (a non-integer, over/underflow, etc.) Write the body of the function. Of course, you should not call any C runtime library function inside the body.

```
int StrToInt(const char* s)
{
  int x = 0;
  int minus = 0;

  if (*s == '-') {   /* check the negative sign symbol */
      minus = 1;
      s++;
   } else if (*s == '+') { /* ignore a positive sign */
      s++;
   }

  while (*s != 0) {
     x = 10 * x + (*s - '0');
     s++;
  }
 return (minus) ? (-x) : (x);
}
```

Name: Student ID:

(2) (10 points) Implement strncpy(char *dest, const char *src, size_t n); which copies n bytes of src to dest. It is similar to strcpy(), except that at most n bytes of src are copied. **Warning:** If there is no null byte among the first n bytes of src, the string placed in dest will not be null-terminated. If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total of n bytes are written. Write the body of the function. Of course, you should not call any C runtime library function inside the body.

```
char* strncpy(char* dest, const char *src, size_t n)
{
	char *p = dest;
	size_t i;

	while (n && *src) {
		*p++ = *src++;
		n--;
	}
	for (i = 0; i < n; i++)
		*p++ = 0;

	return dest;
}
```

Name: Student ID:

3. (20 points) Building a Multi-file Program

We have three C source code files: main.c, plus_one.c, and plus_two.c as follows.

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

int plus_one(int *);
extern int plus_two(int *);
int two = 2;
extern int one;

int main(const int argc, const char **argv)
{
  const int x = atoi(argv[1]);
  int (*ops[3])(int *) = {plus_one, plus_two,};
  int (**pop)(int *) = &ops[0];
  while (**pop != NULL) {
    printf("%d\n", (**pop++)((int *)&x));
  }
  return 0;
}
```

```
/* plus_one.c */
const int one = 1;
int plus_one(int *x)
{
  *x += one;
  return *x;
}
```

```
/* plus_two.c */
const int two = 2;
int plus_two(int *x)
{
  *x += two;
  return *x;
}
```

Name: Student ID:

(1) (5 points) Do we expect any error(s) if we compile the files like ‘gcc209 –c main.c plus_one.c plus_two.c’ in a shell? Describe all errors if any and describe how to fix them.

⇨ No error occurs in compilation.

(2) (5 points) After fixing potential errors in (1), we go on to type ‘gcc209 –o funprog main.o plus_one.o plus_two.o’. Do we expect any errors? Describe all errors if any and describe how to fix them.

⇨ A linkage error: the variable, two, is defined twice in the files (main.c and plus_two.c)

⇨ Remove int two = 2; in main.c since it’s not used in the file

⇨ Or replace int two = 2; in main.c with extern int two;

⇨ Or replace const int two = 2 in plus_two.c with extern int two;

(3) (5 points) After fixing potential errors in (1), (2), we go on to type ‘./funprog’ in a shell. What does it print out?

⇨ It crashes due to memory access violation since it accesses argv[1] which is NULL.

(4) (5 points) What does it print out if we type ‘./funprog 3 5’?

⇨ 4

6

Name: Student ID:

4. (20 points) Reverse the World

(1) (8 points) `reverse_bits(char x)` reverses the order of bits in `x` and returns it. Write the function body below.

```
char reverse_bits(char x)
{
 int i;
 for (i = 0; i < 4; i++) {
   char a = (x & ((0x1) << (7 - i)));
   char b = (x & ((0x1) << i));
   x &= (0xFF ^ ((0x1) << (7-i)) ^ ((0x1) << (i)));
   if (a) x |= ((0x1) << (i));
   if (b) x |= ((0x1) << (7-i));
 }
 return x;
}
```

⇨ There are many correct solutions whose code looks different from the above.

Name: Student ID:

(2) (12 points) `reverse_bytes(char *s, int n)` reverses the order of bits in 8`n-bit number` (s[0], …, s[n-1]) and stores the result in s. That is, the leftmost bit in s[0] should be stored at the rightmost bit position of s[n-1].

```
void reverse_bytes(char *s, int n)
{
  char *b = s, *e = s + (n-1);

  while (b <= e) {
     char t = reverse_bits(*e);
     *e = reverse_bits(*b);
     *b = t;
     b++, e--;
  }
}
```

Name: Student ID:

## 5. (20 points) Describing an Algorithm in Pseudo Code

(1) (10 points) My program is given a text file that holds the content of an entire book, and prints out 100 most popular words (in terms of number of repetitions) in the text of the book. Describe the algorithm of this program. Be as specific as possible such as which data structures to use, etc. Also, the program should be reasonably efficient.

⇨ 1. Read one word, k, at a time

2. Look up k in a key-value hash table

If the node with k is found, increment the value of the node by 1

If not {

add a new node (key=k, value=1) to the hash table

increment the unique word count (a global variable, initialized to 0) by 1

}

3. Repeat 1 and 2 until it processes every word in the book.

4. Allocate an array of pointers of the node whose size equals to the unique node count, like,

```
struct node *parray = malloc (node_count * sizeof(struct node *));
```

5. Traverse the nodes in the hash table, and copy the pointer to each node to parray.

6. Sort the parray by qsort() by the descending order of the value of each node, and print up to 100 nodes from the sorted array.

Name: Student ID:

(2) (10 points) My program is given a text file that contains 30 million 8-digit integers (e.g., 23,980,368) that are unique (no repetition among the numbers), and should sort these numbers in the ascending order. The only problem is that the memory of the computer is very small (just 15 MB of available memory for the program), not even enough for reading all 30 million integers at once. Describe an efficient algorithm that does this task.

⇨ We will use a large bitmap whose size is $10^8$ (which can hold $10^8$ bits). That is, we allocate $10^8/8$ bytes, which requires 12.5 MB of memory.

1. Allocate a char array whose size is 12.5 MB, and initialize them to 0.
2. Read one integer at a time, and set the corresponding bit of the bitmap to 1 by using the integer as an index to the bitmap.
3. Repeat 2 until it processes all integers in the file.
4. for (int = 0; i < $10^8$; i++)

   if (the bitmap for index i is 1) print i;

This requires reading 30 million integers from the file, and scanning each bit in the bitmap ($10^8$ entries).

You can do partial sorting and do external merge sort. For example,

1. read the first 1 million integers, sort them., and write them to sfile1
   A. read the second 1 million integers, sort them, and write them to sfile2,
   B. …
   C. read the last 1 million integers, sort them, and write them to sfile30.
2. Merge sort sfile1 .. sfile30
   A. Open all the files sfile1,..,sfile30
   B. Read one integer respectively from sfile1, …, sfile30
   C. Write the maximum of those 30 intgers to stdout, read one more integer from the file that the written integer came from.
   D. Repeat 2-C until there is no more integer in any of the files

Note that this still works, but it's very inefficient (lots of slow disk I/O and sort operations). So, -3 points for lack of efficiency.