Midterm Exam
Nov 8th, 2012

COMS W3157 Advanced Programming
Columbia University
Fall 2012

Instructor: Jae Woo Lee

About this exam:

- There are 4 problems totaling 100 points:

  problem 1: 30 points
  problem 2: 30 points
  problem 3: 30 points
  problem 4: 10 points

- Your answers should be written directly to the exam booklet itself. Do not use any bluebook. The exam booklet has ample space for your answers. If you need more for some reason, blank sheets of papers will be provided. Make sure to put your name and UNI in all loose sheets you turn in.

- Everything you write will be considered when we grade. This can be good or bad. If you give two different answers to a question, we will take the one that will result in a LOWER grade. So make sure to cross out clearly anything that you don't consider your final answer.

- Assume the following programming environment:

  Language: C
  Compiler: gcc
  Platform: Ubuntu Linux 12.04, 64-bit version
  Primitive type sizes: sizeof(int) is 4 and sizeof(int *) is 8

Please write your name and UNI:

Name: _____

UNI: _____

Good luck!

Problem [1]: 30 points (15 expressions, 2 points each)

Consider a C program's main() function that starts as follows:

```
struct MdbRec {
    char name[16];
    char  msg[24];
};

int main(int argc, char **argv)
{
    int a[10] = { 100, 101, 102, 103, 104, 105, 106, 107, 108, 109 };
    int *p = a + 2;

    struct MdbRec m[1];
    strcpy(m[0].name, "dude");
    strcpy(m[0].msg,  "0");

    int x = 0xffffffff;
    int y = 0xffffffff;
    unsigned char *c = (unsigned char *)&x;
```

Evaluate the 15 given expressions in the context of the main() function. Here is what I mean by evaluating an expression:

- For integer expressions (i.e., the expressions whose types are char, short, int, size_t, long, or long long--either signed or unsigned), write the actual number value in decimal notation.

- For non-integer expressions, write the type name, in the format that you use to declare a variable of that type. Some example type names include (but not limited to):

```
int *
double
double **
int (*)(int)
```

- Write "invalid" if a given expression is not a valid C expression.

Please note that:

- Boolean expressions are integer expressions in C. Please do not write "true" or "false".

- You can assume that the program is run with no command line argument.

- None of the given expressions have const types, i.e., you don't need to worry about putting 'const' in front of any of the types.

Please be careful. Some of these are VERY tricky.

Problem [1] continued

(1.1) p[2] _____

(1.2) sizeof(p) _____

(1.3) m->msg + 1 _____

(1.4) "0"[1]==0 && "0"[1]=='\0' _____

(1.5) *argv++ _____

(1.6) m->msg - (char *)m _____

(1.7) &m[0] _____

(1.8) sizeof(m[0].msg) _____

Problem [1] continued

(1.9)	&p	_____
(1.10)	x + y	_____
(1.11)	c[2]	_____
(1.12)	&c	_____
(1.13)	(~x) & y	_____
(1.14)	sizeof(a) / sizeof(a[0])	_____
(1.15)	*&*&a[5]	_____

Problem [2]: 30 points

Consider the following program, arg.c, shown with line numbers:

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <stdlib.h>
 4
 5  int main(int argc, char **argv)
 6  {
 7      if (argc != 3) {
 8          printf("usage: %s <arg1> <arg2>\n", argv[0]);
 9          return 1;
10      }
11
12      char *a = *++argv;
13      char *b = *++argv;
14      char c[2] = { 0, 0 };
15      char *p;
16      char *q;
17
18      for (p = a; *p; p++) {
19          if ('a' <= *p && *p <= 'z')
20              *c = *p + 'A' - 'a';
21          else
22              *c = *p;
23          printf("%s", c);
24      }
25
26      p = (char *)malloc(strlen(a) + strlen(b) + 1);
27      strcpy(p, a);
28      q = p + strlen(a);
29      while ((*q++ = *b++) != 0)
30          ;
31      printf("%s\n", p);
32      p = NULL;
33      return 0;
34  }
```

It is compiled, linked and run with two command line arguments as follows:

```
$ gcc -g -Wall arg.c
$ ./a.out abc 123
```

Answer the following questions:

Problem [2] continued

(a) 15 points

What is the output? Assume that the malloc() call was successful. If you think that the program contains bugs (other than memory leaks) that can make the program crash (ex. segmentation fault), write "CAN CRASH" and then write the most likely output if the program happens to run without crashing.

(b) 5 points

If the program is run using valgrind, how many bytes will valgrind report as "definitely lost"? You can write 0 if you think there is no memory leak that valgrind will consider "definitely lost".

Please make sure to write the number of BYTES.

Problem [2] continued

(c) 5 points

Identify the line numbers that contains memory errors other than memory leaks (invalid access, for example.) If you think that there is no memory error other than possible memory leaks, write NONE.

You don’t have to identify the nature of the memory errors. Just line numbers.

(d) 5 points

Modify the program to fix ALL memory errors, including memory leaks. Write only the lines that need to be fixed. Write the line number and the line of code that will replace that line. If you think there is nothing to fix, write "NO CHANGE".

The program can be memory error free by modifying 0 to 3 lines of code. Please do not modify more than 3 lines of code.

Problem [3]: 30 points

Consider a mdb database, my-mdb, containing 2 records. We ran mdb-lookup program on it to display the records:

```
$ ./mdb-lookup  my-mdb
lookup:
   1: {Obama} said {likable enough Hillary}
   2: {Romney} said {binders full of women}

lookup:
```

Consider the following program, mdb-print.c, which is supposed to display all the records in a given database without taking any user input. It is supposed to do the same thing that the mdb-lookup does when you just press ENTER at the lookup prompt (except that mdb-print will not display the line numbers.)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include "mylist.h"
 4
 5  struct MdbRec {
 6      char name[16];
 7      char  msg[24];
 8  };
 9
10  int main(int argc, char **argv)
11  {
12      FILE *fp = fopen(argv[1], "rb");
13
14      struct List list;
15      initList(&list);
16
17      struct Node *node = NULL;
18      struct MdbRec *p;
19
20      p = (struct MdbRec *)malloc(sizeof(struct MdbRec));
21      while (fread(p, sizeof(struct MdbRec), 1, fp) == 1) {
22          node = addAfter(&list, node, p);
23      }
24
25      while ((p = (struct MdbRec *)popFront(&list)) != NULL) {
26          printf("{%s} said {%s}\n", p->name, p->msg);
27      }
28
29      fclose(fp);
30      return 0;
31  }
```

Suppose that you compiled, linked, and ran the mdb-print program with my-mdb argument. Assume that the program ran without any I/O error, that all malloc() calls succeed, and that there is no memory error other than memory leaks. The program ran without crashing. (I removed all error checking code to make the program easier to read. Also, a note on using fread() and an excerpt from mylist.h is included at the end of this problem.)

Problem [3]: continued

The program, however, may or may not contain memory leaks, and also, may or may not produce the desired output.

(a) 10 points

What is the output? If you think it will produce the desired output (i.e. same as mdb-lookup when you press ENTER, except the line numbers), then simply write "SAME AS MDB-LOOKUP".

(b) 5 points

If the program is run using valgrind, how many bytes will valgrind report as "definitely lost"? You can write 0 if you think there is no memory leak that valgrind will consider "definitely lost".

Please make sure to write the number of BYTES.

Problem [3]: continued

(c) 15 points

The program, as I said, may or may not contain memory leaks, and also, may or may not produce the desired output.

Modify the program so that it produces the desired output and it contains no memory leaks. Write "NO CHANGE" if you think no modification is needed.

There is another requirement for making modifications. You can only add new lines of code. You cannot modify or remove any existing lines of code. For each line you add, clearly identify the two existing line numbers that your new line will go in between.

The program can be corrected by adding 0 to 5 lines of code. Please do not add more than 5 lines of code.

```
// How to use fread():

/* size_t fread(void *p, size_t size, size_t n, FILE *file)
 *
 * - reads 'n' objects, each 'size' bytes long, from 'file'
 *   into the memory location pointed to by 'p'.
 *
 * - returns the number of objects successfully read, which may be
 *   less than the requested number n if the end of file has been
 *   reached or an I/O error has occurred.
 */

// An excerpt from mylist.h:

struct Node {
    void *data;
    struct Node *next;
};

struct List {
    struct Node *head;
};

static inline void initList(struct List *list) {
    list->head = 0;
}

/* Create a node that holds the given data pointer,
 * and add the node to the front of the list.
 *
 * Note that this function does not manage the lifetime of the object
 * pointed to by 'data'.
 *
 * It returns the newly created node on success.
 * It returns NULL if malloc fails.
 */
struct Node *addFront(struct List *list, void *data);

/* Remove the first node from the list, deallocate the memory for the
 * ndoe, and return the 'data' pointer that was stored in the node.
 * Returns NULL is the list is empty.
 */
void *popFront(struct List *list);

/* Create a node that holds the given data pointer,
 * and add the node right after the node passed in as the 'prevNode'
 * parameter.  If 'prevNode' is NULL, this function is equivalent to
 * addFront().
 *
 * Note that this function does not manage the lifetime of the object
 * pointed to by 'data'.
 *
 * It returns the newly created node on success.
 * It returns NULL if malloc fails.
 */
struct Node *addAfter(struct List *list,
                      struct Node *prevNode, void *data);
```

Problem [4]: 10 points

Consider the following program, sum55.c:

```
    #include <stdio.h>

    int f()
    {
        // SOME MAGICAL CODE GOES HERE...









    }

    int main()
    {
        int i;
        int sum = 0;

        for (i = 0; i < 10; i++)
            sum += f();

        printf("%d\n", sum);
        return 0;
    }
```

Implement the body of the function f() so that when this program is compiled, linked, and run, it will output "55" as follows:

```
    $ gcc -g -Wall sum55.c
    $ ./a.out
    55
```

You can only implement the body of f. You CANNOT change the prototype of f. You CANNOT add any other function. You CANNOT call any library function from the body of f.

[blank page]

[blank page]

[blank page]