Midterm Exam #2
Apr 14, 2016

COMS W3157 Advanced Programming
Columbia University
Spring 2016

Instructor: Jae Woo Lee

About this exam:

- There are 4 problems totaling 100 points:

  Problem 1: 36 points
  Problem 2: 14 points
  Problem 3: 30 points
  Problem 4: 20 points

- Assume the following programming environment:

  All programs are built and run on Ubuntu Linux 12.04, 64-bit version, where sizeof(int) is 4 and sizeof(int *) is 8.

  All library function calls and system calls are successful. For example, you can assume malloc() does not return NULL.

  For all program code in this exam, assume that all the necessary #include statements are there even if they are not shown.

  If this exam refers to lab code, assume the versions provided by Jae, i.e., skeleton code and solutions.

  When you are asked to write code, you should avoid using hardcoded numbers as much as possible. Hardcoded numbers make your program error prone, less extensible, and less portable. For example, using "sizeof(int *)" instead of "8" will make it correct for both 32-bit and 64-bit environments.

What to hand in and what to keep:

- At the end of the exam, you will hand in only the answer sheet, which is the last two pages (one sheet printed double-sided) of this exam booklet.

- Make sure you write your name & UNI on the answer sheet.

- All other pages (i.e., the rest of this exam booklet and any blue book you used during the exam) are yours to keep.

- Before you hand in your answer sheet, please copy down your answers back onto the exam booklet so that you can verify your grade when the solution is published in the mailing list.

Good luck!

```
+-------------------------------------------------------------------+
|  PLEASE DO NOT OPEN THIS EXAM BOOKLET UNTIL YOU ARE TOLD TO DO SO!  |
+-------------------------------------------------------------------+
```

Problem [1]: 36 points total, 12 parts, 3 points each

Please write "True" or "False" for each of the 12 statements.

(1.1) The fdopen() function takes an existing file descriptor as one of its arguments, and returns a pointer of type FILE*.

(1.2) In C, unlike a local variable, a variable declared with "static" keyword is shared between parent and child processes.

(1.3) Unlike a stack-allocated object, a heap-allocated object is shared between parent and child processes.

(1.4) In C, malloc(sizeof(struct Pt)) simply allocates sizeof(struct Pt) bytes on the heap, but in C++, malloc(sizeof(Pt)) first allocates sizeof(struct Pt) bytes on the heap and then calls Pt::Pt(), the default constructor.

For (1.5) - (1.8), consider the following netcat process currently executing in a pipeline:

```
program_one | nc -l 4444
```

Assume that a TCP connection between this netcat process and a remote netcat process has been established.

(1.5) This netcat process is running in server mode.

(1.6) This netcat process will copy bytes it receives from the TCP connection back out to the same TCP connection.

(1.7) The bytes that the program_one writes to standard output will be sent to the remote netcat process.

(1.8) This netcat process will copy bytes it receives from the standard input directly to standard output.

For (1.9) - (1.12), consider fork() and execl() functions. Here are their prototypes from the man pages:

```
pid_t fork(void);

int execl(const char *path, const char *arg, ...);
```

Both functions return -1 on failure.

(1.9) fork() creates a new process by duplicating the calling process.

(1.10) The value of the expression (getpid() == fork()) is always 0.

(1.11) The value of the expression (fork() == fork()) is always 0.

(1.12) The execl() function returns 0 on success.

Problem [2]: 14 points total, 4 parts

Consider the following C program:

```
    void str_dup(  ____________________________  t, char *s)  // (2.1)
    {                                                         //  3 points

        char *p;

        ____________________________ = p = malloc(strlen(s) + 1);  // (2.2)
                                                              //  3 points


        while ((  ____________________________  ) != 0) {}  // (2.3)
    }                                                         //  5 points


    int main()
    {
        char *a;

        str_dup(  ____________________________  , "XYZ");  // (2.4)
                                                          //  3 points

        printf("%s\n", a);

        free(a);

        return 0;
    }
```

Complete the program code by filling in the blanks, so that the program will build without any error or warning, run without any memory error or leaks, and prints XYZ, as shown below:

```
    $ gcc -g -Wall dup.c
    $ ./a.out
    XYZ
```

Each blank is an expression or a type name in C, and does NOT contain any of the following characters:

```
    } { ) ( ; ,
```

That is, do NOT put any curly braces, parentheses, semicolons or commas.

Make sure to write your answers on your answer sheet.

[This problem is identical to the problem [2] of midterm #1.]

Problem [3]: 8 parts, 30 points total

(3.1) 14 points

Write the output of the following C++ program, loop.cpp:

```
struct X {
    int n;
    X(int a) {
        n = a;
    }
    ~X() {
        if (n)
            printf("%d,", n);
    }
};

struct Y {
    int n;
    Y(int a, X x) {
        n = a;
        x.n = 0;
    }
    ~Y() {
        if (n)
            printf("%d,", n);
    }
};

int main() {
    int i, j;
    for (i = 1; i <= 2; i++) { // i goes from 1 to 2
        X x(i);
        for (j = 3; j <= 4; j++) { // j goes from 3 to 4
            Y y(j, x);
        }
    }
    printf("\n");
    return 0;
}
```

Problem [3] continued

Now, consider the following C program, loop.c, which tries to emulate the behavior of the previous C++ program loop.cpp. Fill in the blanks in loop.c so that the execution of the C program mimics the C++ program as closely as possible. One way to think about it is that loop.c is the output when you gave loop.cpp as an input to a translator that converts C++ code to C code.

```
struct X { int n; };

void Xc( ____________________________________ )   // (3.2)
{
    this->n = a;
}

void Xd( ____________________________________ )   // (3.3)
{
    if (this->n)
        printf("%d,", this->n);
}

struct Y { int n; };

void Yc( ____________________________________ )   // (3.4)
{
    this->n = a;
    x.n = 0;
}

void Yd( ____________________________________ )   // (3.5)
{
    if (this->n)
        printf("%d,", this->n);
}

int main() {
    int i, j;
    for (i = 1; i <= 2; i++) {
        struct X x;

        ____________________________      // (3.6)

        for (j = 3; j <= 4; j++) {
            struct Y y;

            ____________________________      // (3.7)

        }

        ____________________________      // (3.8)

    }
    printf("\n");
    return 0;
}
```

For (3.6), (3.7) and (3.8), write zero, one, or two C statements.

Problem [4]: 20 points

Consider a simplified version of the "cat" program, cat1.c, which takes a single file name as an argument, and copies the file to the standard output.

Here is an example shell session building and running the program:

```
$ gcc -Wall cat1.c -o cat1
$ ./cat1 ap-midterm.pdf > copy
$ cmp ap-midterm.pdf copy
$ ll ap-midterm.pdf copy
-rw-r--r--  1 jae  phd  11170 Apr 13 19:49 ap-midterm.pdf
-rw-r--r--  1 jae  phd  11170 Apr 13 19:51 copy
```

The cat1.c file is partially given below. Complete the code, following the requirements listed in the comments.

```
static void die(const char *msg) { perror(msg); exit(1); }

int main(int argc, char **argv)
{
    size_t num;
    char   buf[4096];

    if (argc != 2) {
        fprintf(stderr, "usage: %s <filename>\n", argv[0]);
        exit(1);
    }

    FILE *fp = fopen(argv[1], "rb");
    if (fp == NULL)
        die(argv[1]);

    //  Your code begins here
    /*  Requirements:
        - Do NOT declare any additional variables.
        - Do NOT assume that the given file is a text file.
        - Minimize the number of calls to library functions. */


    //  WRITE YOUR CODE ON THE ANSWER SHEET.


    //  Your code ends here

    if (ferror(fp))
        die("file operation failed");

    fclose(fp);
    return 0;
}
```

Problem [4]: continued

[Excerpt from lecture note]

char *fgets(char *buffer, int size, FILE *file)

- reads at most size-1 characters into buffer, stopping if newline is read (the newline is included in the characters read), and terminating the buffer with '\0'

- returns NULL on EOF or error (you can call ferror() afterwards to find out if there was an error).

- Never use the similar gets() function; it's UNSAFE!

int fputs(const char *str, FILE *file)

- writes str to file.

- returns EOF on error.

int fscanf(FILE *file, const char *format, ...)
int fprintf(FILE *file, const char *format, ...)

- file I/O version of scanf & printf

fclose(FILE *file)

- closes the file

int fseek(FILE *file, long offset, int whence)

- Sets the file position for next read or write. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by whence. If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

- returns 0 on success, non-zero on error

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, in which case feof() and ferror() can be used to determine status.

size_t fwrite(const void *p, size_t size, size_t n, FILE *file)

- writes n objects, each size bytes long, from the memory location pointed to by p out to file.

- returns the number of objects successfully written, which will be less than n when there is an error.

[blank page]

Name: _____

UNI: _____

[2]

(2.1) 3 points: _____ t,

(2.2) 3 points: _____ = p = ...

(2.3) 5 points: (_____) != 0

(2.4) 3 points: _____ , "XYZ"

[4]

```
//  Your code begins here
/*  Requirements:
    - Do NOT declare any additional variables.
    - Do NOT assume that the given file is a text file.
    - Minimize the number of calls to library functions. */
```

```
//  Your code ends here
```

Name: _____ | left->UNI: _____

UNI: _____ | right->UNI: _____

[1]

(1.1) _____ | (1.5) _____ | (1.9) _____

(1.2) _____ | (1.6) _____ | (1.10) _____

(1.3) _____ | (1.7) _____ | (1.11) _____

(1.4) _____ | (1.8) _____ | (1.12) _____

(Problem [2] & [4] are on the other side of this sheet.)

[3]

(3.1) _____

(3.2) void Xc(_____)

(3.3) void Xd(_____)

(3.4) void Yc(_____)

(3.5) void Yd(_____)

(3.6) _____

(3.7) _____

(3.8) _____