Final Exam May 9, 2017

COMS W3157 Advanced Programming Prof. Jae Woo Lee
Spring 2017 Columbia University

---

About this exam:

- There are 3 problems totaling 100 points:

  Problem 1: 54 points
  Problem 2: 26 points
  Problem 3: 20 points

- Assume the following programming environment:

  All programs are built and run on Ubuntu Linux 16.04, 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 writing code, 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. You keep the rest of the exam booklet.

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

- Please write only your final answers on the answer sheet. Verbosity will only hurt your grade because, if we find multiple answers to a question, we will cherry-pick the part that will result in the LOWEST grade. This policy ensures that a shotgun approach to solving a problem is never rewarded. Please make sure you cross out clearly anything that you don’t consider your final answer.

- 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!  |
+-------------------------------------------------------------------+
```

References: SmartPtr template class definition
---

```
template <class T>
class SmartPtr {

    private:

        T *ptr;

        int *count;

    public:

        explicit SmartPtr(T *p = 0)
        {
            ptr = p;
            count = new int(1);
        }

        SmartPtr(const SmartPtr<T>& sp)
            : ptr(sp.ptr), count(sp.count)
        {
            ++*count;
        }

        ~SmartPtr()
        {
            if (--*count == 0) {
                delete count;
                delete ptr;
            }
        }

        SmartPtr<T>& operator=(const SmartPtr<T>& sp)
        {
            if (this != &sp) {
                // first, detach.
                if (--*count == 0) {
                    delete count;
                    delete ptr;
                }
                // attach to the new object.
                ptr = sp.ptr;
                count = sp.count;
                ++*count;
            }
            return *this;
        }

        T& operator*() const { return *ptr; }

        T* operator->() const { return ptr; }

        T* getPtr() const { return ptr; }

        operator void*() const { return ptr; }
};
```

References
----------

For all C++ programs, assume the following declarations and definitions:

```
    template <typename T>
    struct SmartPtrStruct {
        T   *ptr;
        int *count;
    };

    // Returns the current reference count of the given SmartPtr
    template <typename T>
    int ref_count(const SmartPtr<T>& p) {
        // A hack to access count, which is a private member
        return *(((SmartPtrStruct<T> *)&p)->count);
    }

    /*
     * Assume the classify_pointer function exists, is correct, doesn't
     * leak memory, and works as follows:
     *
     *   It prints "STACK" if p contains an address on the stack;
     *   it prints "HEAP" if p contains an address on the heap;
     *   it prints "NEITHER" if p contains an address neither on the stack
     *                                                 nor on the heap.
     */
    void classify_pointer(const void *p);

    template <typename T>
    void print(T t) { cout << t << endl; }
```

---

```
void *memcpy(void *dest, const void *src, size_t n);

    The memcpy() function copies n bytes from memory area src to memory
    area dest.  The memory areas should not overlap.  The memcpy()
    function returns a pointer to dest.
```

---

Problem [1]: 54 points total, 18 parts, 3 points each
---
Consider the following C++ program:

```
    int main()
    {
        static int i;
        MyString s[3];  s[0] = "0";  s[1] = "1";  s[2] = "2";

        vector< MyString * >          v1;
        vector< MyString >            v2;
        vector< MyString * >          v3;
        vector< SmartPtr<MyString> >  v4;

        for (i = 0; i < 3; i++) { v1.push_back( s + i               ); }
        for (i = 0; i < 3; i++) { v2.push_back( *v1[i]              ); }
        for (i = 0; i < 3; i++) { v3.push_back( new MyString(v2[i]) ); }
        for (i = 0; i < 3; i++) {
            SmartPtr<MyString> sp(v3[i]);
            v4.push_back(sp);
        }

        cout << "\n(1.1) "; classify_pointer( "0" );

        cout << "\n(1.2) "; classify_pointer( &i );

        cout << "\n(1.3) "; classify_pointer( s );

        cout << "\n(1.4) "; classify_pointer( v1[0] );

        cout << "\n(1.5) "; classify_pointer( &v1[0] );

        cout << "\n(1.6) "; classify_pointer( &v1 );

        cout << "\n (1.7) "; print((int)( &s[0] ==  v1[0] ));

        cout << "\n (1.8) "; print((int)( &s[0] == &v2[0] ));

        cout << "\n (1.9) "; print((int)(  s[0] ==  v2[0] ));

        cout << "\n(1.10) "; print((int)( v2[0] == *v3[0] ));

        SmartPtr<MyString> *p3 = new SmartPtr<MyString>( v3[0] );
        SmartPtr<MyString> *p4 = new SmartPtr<MyString>( v4[0] );

        cout << "\n(1.11) "; classify_pointer( &p4 );

        cout << "\n(1.12) "; classify_pointer( *p4 );

        cout << "\n(1.13) "; classify_pointer( p4 );

        cout << "\n(1.14) "; print( ref_count(v4[0]) );

        cout << "\n(1.15) "; print( ref_count(v4[1]) );

        cout << "\n(1.16) "; print((int)( v2[1][1] ));

        /* (1.17) - (1.18): Questions on memory leak */          return 0; }
```

Problem [1] continued
---

(1.1) - (1.16)

The program builds without error, runs without crashing, and successfully produces 16 lines of output for (1.1) ... (1.16). Fill in the blanks on the answer sheet to match the program's output.

(1.17)

The program may or may not leak memory. How many SmartPtr objects and how many MyString objects are leaked at the end of the program? Write the number of objects, not the number of bytes. (Write 0 for no leak.)

(1.18)

How many total bytes are leaked at the end of the program? Write the total number of bytes lost, as reported by Valgrind. (Write 0 for no leak.)

Note that sizeof(MyString) is 16 in our system due to 4-byte padding.

---

Problem [2]: 26 points total, 4 parts
---

(2.1) 8 points

Consider the following C++ program:

```
    struct ListHead { struct ListHead *next; };

    void free_list(ListHead *l) { if (l) { free_list(l->next); free(l); } }

    template <typename T>
    void print_list(ListHead *list, T *dummy) {
        while (list) {
                                        /* Fill in the blank */
            cout << '\t' << *(  ____________________________________________  );

            list = list->next;
        }
        cout << endl;
    }

    int main() {
        ListHead *list = NULL;
        for (int i = 0; i < 3; i++) {
            ListHead *p = (ListHead *)malloc(sizeof(ListHead) + sizeof(i));
            memcpy(p + 1, &i, sizeof(i));
            p->next = list; list = p;
        }
        print_list(list, (int *)NULL); free_list(list);

        list = NULL;
        for (double d = 50.0; d < 300.0; d += 100.0) {
            ListHead *p = (ListHead *)malloc(sizeof(ListHead) + sizeof(d));
            memcpy(p + 1, &d, sizeof(d));
            p->next = list; list = p;
        }
        print_list(list, (double *)NULL); free_list(list);
        return 0;
    }
```

Fill in the blank on the answer sheet with a SINGLE C++ EXPRESSION so that the program produces the following output:

```
    $ g++ -Wall list-head.cpp && ./a.out
            2       1       0
            250     150     50
```

---

(2.2) 6 points

Fill in the blanks on the answer sheet to complete the prototype of the fdopen() library function:

```
    /* return type */           /* argument type */
    ________________  fdopen(  ____________________  arg, const char *mode);
```

Problem [2] continued
---

(2.3) 6 points

Write the output of the following C program:

```
int main()
{
    int *a = malloc(4 * sizeof(int));
    a[0] = a[1] = a[2] = a[3] = 0;

    pid_t pid = fork();
    if (pid == 0) { // child process
        a[1] = a[3] = 1;
    } else { // parent process
        waitpid(pid, NULL, 0); // no status and no options
        printf("%d,%d,%d,%d\n", a[0],a[1],a[2],a[3]);
    }
    return 0;
}
```

---

(2.4) 6 points

Consider the following C++ program:

```
struct X {
    X(int n) {
        n--;
        fprintf(stderr, "%d", n);

        char argv1[100];
        sprintf(argv1, "%d", n);
        execl("./a.out", "a.out", argv1, (char *)0);
    }
    ~X() {
        fprintf(stderr, "X");
    }
};

int main(int argc, char **argv)
{
    int n = atoi(argv[1]);
    if (n) {
        X x(n);
        fprintf(stderr, "Y");
    }
    fprintf(stderr, "\n");
    return 0;
}
```

Write the output when the program is run with an argument "4" as follows:

```
$ g++ -Wall exec-test.cpp
$ ./a.out 4
```

Problem [3]: 20 points total, 5 parts, 4 points each
---

(3.1)

Which of the following Socket API functions is NOT called by
mdb-lookup-server.c from lab6?

(A) bind
(B) listen
(C) accept
(D) connect
(E) None of the above -- i.e., mdb-lookup-server.c calls all of them

---

(3.2)

Which of the following Socket API functions is NOT called by
http-server.c from lab7?

(A) bind
(B) listen
(C) accept
(D) connect
(E) None of the above -- i.e., http-server.c calls all of them

---

(3.3)

Which of the following shell commands does NOT work as an echo server
listening on port 50000? (Assume that mypipe is a named pipe created by
"mkfifo mypipe". Also note that the "cat" command will read bytes from
standard input when you give "-" as an argument instead of a file name.)

(A) cat mypipe | nc -l 50000 > mypipe
(B) cat - < mypipe | nc -l 50000 > mypipe
(C) nc -l 50000 < mypipe | cat - > mypipe
(D) cat - < mypipe | nc -l 50000 | cat - > mypipe
(E) None of the above -- i.e., all of the above work as an echo server
listening on port 50000.

---

(3.4)

The Internet Protocol (IP) only provides best-effort delivery and its
service is characterized as unreliable. (True / False)

---

(3.5)

The vector template class in the C++ standard library ensures that each
push_back() operation is performed in O(1) time. (True / False)

COMS W3157, Spring 2017: Final Exam Answer Sheet, page 1 of 2

UNI: Name:

---

[2]

(2.1) cout << '\t' << *( _____ );

(2.2) _____ fdopen( _____ arg, const char *mode);

(2.3)
fork: _____

(2.4)
exec: _____

[3]

(3.1) A/B/C/D/E: _____

(3.2) A/B/C/D/E: _____

(3.3) A/B/C/D/E: _____

(3.4) True/False: _____

(3.5) True/False: _____

COMS W3157 Spring 2017
Final Exam
Answer Sheet, page 2 of 2

Your UNI: +---+---+---+---+---+---+---+---+
| | | | | | | | |
+---+---+---+---+---+---+---+---+

Your Name: _____

front->UNI: _____

left->UNI: _____ [You] right->UNI: _____

back->UNI: _____

[1]

(1.1) _____

(1.2) _____

(1.3) _____

(1.4) _____

(1.5) _____

(1.6) _____

(1.7) _____

(1.8) _____

(1.9) _____

(1.10) _____

(1.11) _____

(1.12) _____

(1.13) _____

(1.14) _____

(1.15) _____

(1.16) _____

(1.17) _____ SmartPtr objects

and _____ MyString Objects

(1.18) _____ bytes total