```
Final Exam                                                    Dec 22, 2016

COMS W3157 Advanced Programming                           Prof. Jae Woo Lee
Fall 2016                                               Columbia University
```

About this exam:

- There are 3 problems totaling 100 points:

      Problem 1: 54 points
      Problem 2: 36 points
      Problem 3: 10 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
----------

FILE *fopen(const char *filename, const char *mode)

  - Opens a file, and returns a FILE* that you can pass to other
    file-related functions to tell them on which file they should operate.

  - "r" open for reading (file must already exist)
    "w" open for writing (will trash existing file)
    "a" open for appending (writes will always go to the end of file)

    "r+" open for reading & writing (file must already exist)
    "w+" open for reading & writing (will trash existing file)
    "a+" open for reading & appending (writes will go to end of file)

  - returns NULL if file could not be opened for some reason

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.

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).

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

  - writes str to file.
  - returns EOF on error.
```

```
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; }
};
```

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

    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; }

    // sp-test.cpp continues on the next page
```

```
(1.1) - (1.14)

The program successfully produces 14 lines of output for (1.1) ... (1.14),
but it may or may not crash at the end, and it may or may not leak memory.

Fill in the blanks on the answer sheet to match the program’s output.
```

```
(1.15) - (1.18): Now suppose we ran the program under Valgrind as follows:

    $ valgrind --leak-check=yes ./a.out

Answer TRUE or FALSE for (1.15) - (1.17), and give a number for (1.18).

(1.15) Valgrind reports "Invalid free() / delete / delete[] / realloc()".
(1.16) Valgrind reports "Invalid read of size 1".
(1.17) Valgrind reports "Conditional jump or move depends on uninitialised
                         value(s)".

(1.18) How many bytes does the program leak?  (Write 0 for no leak.)

           Note that sizeof(MyString) and sizeof(UrString) are 16 in our
           system due to 4-byte padding.  Assume that sizeof(MyString)==16
           && sizeof(UrString)==16 in all your calculations.
```

```
Problem [1]: sp-test.cpp continued from the previous page
----------------------------------------------------------
struct UrString {
    const char *data;
    int         len;

    UrString(const char *s)  { assert(s); len = strlen(s); data = s; }

    const char& operator[](int i) const  { return data[i]; }
};

int main() {

    const char *c = "ABC";  MyString s1(c);  UrString s2(c);

    cout << "\n(1.1) "; classify_pointer(&s2);

    cout << "\n(1.2) "; classify_pointer(c);

    cout << "\n(1.3) "; classify_pointer(&s1[0]);

    cout << "\n(1.4) "; classify_pointer(&s2[0]);

    cout << "\n(1.5) "; print( (int)(c == &s1[0]) );

    cout << "\n(1.6) "; print( (int)(c == &s2[0]) );

    SmartPtr<MyString>  p0;
    SmartPtr<MyString>  p1(new MyString(s1));
    SmartPtr<MyString>  p2(new MyString("XYZ"));

    p0 = p1 = p2;

    cout << "\n(1.7) ";  print( *p0 );

    cout << "\n(1.8) ";  print( ref_count(p0) );

    cout << "\n(1.9) ";  print( *p2 );

    cout << "\n(1.10) "; print( ref_count(p2) );

    cout << "\n(1.11) "; classify_pointer(p2);

    cout << "\n(1.12) "; classify_pointer(&p2);

    SmartPtr<UrString>  p3(new UrString( &(*p2)[0] ));
    SmartPtr<UrString>  p4(p3);

    cout << "\n(1.13) "; print( ref_count(p4) );

    cout << "\n(1.14) "; print( (*p4)[0] );

    // (1.15) - (1.18): Questions on Valgrind output

    // flush buffered output, if any, in case we crash at the end
    fflush(stdout);
    return 0;
}
```

```
Problem [2] 36 points total, 9 parts
-------------------------------------
Recall the mdb-select program from the midterm exam #2. Here is a part of
the shell session that was shown in midterm #2, to refresh your memory:

    jae@clac ~/tmp $ ./mdb-lookup mdb-cs3157
    lookup: AP
     375: {Kanjae} said {I miss the old AP}
    2328: {AP student} said {I'm lonely}
    2407: {confused AP stu} said {but how did you do that}
    3238: {Mark} said {Hello AP!}
    3246: {A} said {Hey AP!!}
    3248: {Paul} said {Hello dudes of AP!}
    3276: {Jill Shah} said {hello AP!}
    3280: {Nick} said {Hello AP'ers}
    3297: {Melanie} said {Hello AP!}
    3306: {Chi} said {Good luck APers!}
    3345: {Donald} said {Make AP great again}
    3391: {m m} said {Thank u APr fantastic!}
    3397: {tranquillo} said {death by AP [C [C [C [C}
    3410: {AP} said {your gpa belong   to me}
    3467: {AP student} said {Just started Lab 4}
    3491: {Darren Hakimi} said {AP with Jae is great}
    4004: {RandomStudent} said {hello AP!}
    4100: {william} said {hello AP}
    4123: {Jordan} said {I'm not in AP yet}

    jae@clac ~/tmp $ ./mdb-select mdb-cs3157 mdb-fun 2328 2407 3345 3410

    jae@clac ~/tmp $ ./mdb-lookup mdb-fun
    lookup:
       1: {AP student} said {I'm lonely}
       2: {confused AP stu} said {but how did you do that}
       3: {Donald} said {Make AP great again}
       4: {AP} said {your gpa belong   to me}
```

(2.1) - (2.5): Fill in the blanks in mdb-select.c on the answer sheet.

Note the following requirements:

- The record numbers on the command line can be in any order -- i.e., they don't have to be in sorted order -- and the records are added to the output database in the specified order.

- Note that the mdb-lookup program prints record numbers starting from 1 -- that is, the very first mdb record in the database file has the record number 1, not 0.

- Do NOT write error checking code. Do NOT declare new variables.

- Every blank space is either a single C/C++ expression or a type name. There should be no semi-colon (;), no curly braces ({}), and no logical operators (&&, ||) within a blank.

- Make sure to get the syntax right. There is NO PARTIAL CREDIT.

The coding rules apply to all parts of this exam unless specified otherwise.

```
Problem [2] continued
---------------------
Now consider mdb-select.cpp, a re-write of mdb-select.c in C++. Fill in the
blanks on the answer sheet so that mdb-select.cpp program runs without any
error or leaks, and produces the same results as the C version.

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

    struct MdbSelector {
        FILE *in_mdb;
        FILE *out_mdb;

        // we disable copy constructor and operator=() - code not shown

        MdbSelector(const char *in, const char *out) {
            in_mdb = fopen(in, "rb"); out_mdb = fopen(out, "ab"); }

        ˜MdbSelector() { fclose(out_mdb); fclose(in_mdb); }

        // (2.6) Fill in the return type

        _________________________________  f(int mdb_rec_num)
        {
            MdbRec R;

            // (2.7) One or two expressions (i.e., one blank could be empty)

            _______________________________________________________________ ;

            _______________________________________________________________ ;

            return __________________________________ ;  // (2.8)
        }

        void g(const MdbRec& R)
        {
            // (2.9) One or two expressions (i.e., one blank could be empty)

            _______________________________________________________________ ;

            _______________________________________________________________ ;
        }
    };
    int main(int argc, char **argv) {
        if (argc < 3) { fprintf(stderr,
            "usage: %s <in-mdb> <out-mdb> [mdb recs]\n", argv[0]); exit(1);
        }

        int i;  vector<MdbRec> v;

        SmartPtr<MdbSelector>  m( new MdbSelector(argv[1], argv[2]) );

        for (i = 3; argv[i]; ++i) { v.push_back( m->f( atoi( argv[i] ))); }

        for (i = 0; i < v.size(); ++i) { m->g( v[i] ); }

        return 0;
    }
```

```
Problem [3] 10 points
----------------------
```

Consider the following C program, eeexec.c:

```
    void eeexec(int n)
    {
        if (n > 0) {

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

            eeexec(n - 1);

            char buf[100];
            sprintf(buf, "%d", n - 1);
            execl("./a.out",  // what file to execute
                  "./a.out",  // argv[0]
                  buf,        // argv[1]
                  (char *)0); // NULL

            printf("%d\n", n);
        }
    }

    int main(int argc, char **argv)
    {
        assert(argc == 2); // one command line argument
        eeexec(atoi(argv[1]));
        return 0;
    }
```

The program is built and run as follows:

```
    gcc -g -Wall eeexec.c

    ./a.out 4
```

Write the output on the answer sheet.

```
COMS W3157, Fall 2016: Final Exam Answer Sheet, page 1 of 2
----------------------------------------------------------------------------
Problem [2]:
///////////////////////////// mdb-select.c //////////////////////////////////

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

int main(int argc, char **argv) {
    if (argc < 3) { fprintf(stderr,
            "usage: %s <in-mdb> <out-mdb> [mdb recs]\n", argv[0]); exit(1);
    }

    FILE *in_mdb;   FILE *out_mdb;   int mdb_rec_num;   struct MdbRec R;

    // Omit error-checking code. Do NOT declare any additional variables.

    in_mdb  = fopen(argv[1], "rb");  out_mdb = fopen(argv[2], "ab");


    for (argv += 3; __________________________________ ; argv++) {  // (2.1)


        mdb_rec_num = atoi( __________________________________ );   // (2.2)
/*
(2.3)
 */     ____________________________________________________________ ;
/*
(2.4)
 */     ____________________________________________________________ ;
/*
(2.5)
 */     ____________________________________________________________ ;
    }
    fclose(out_mdb); fclose(in_mdb); return 0;
}
/////////////////////////// mdb-select.cpp ////////////////////////////////


(2.6) _________________________________  f(int mdb_rec_num)

(2.7) One or two expressions (i.e., one blank could be empty)


    ______________________________________________________________________ ;


    ______________________________________________________________________ ;


(2.8) return __________________________________ ;

(2.9) One or two expressions (i.e., one blank could be empty)


    ______________________________________________________________________ ;


    ______________________________________________________________________ ;
```

```
COMS W3157 Fall 2016                      +---+---+---+---+---+---+---+---+
Final Exam                      Your UNI: |   |   |   |   |   |   |   |   |
Answer Sheet, page 2 of 2                 +---+---+---+---+---+---+---+---+

Your Name:   _______________________________________________________________

                  left->UNI: ________________   right->UNI: ________________


Problem [1]                                  | (Problem [2] on the other side)
                                             |
(1.1) ________________________________       |
                                             |  Problem [3] Write the output:
(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) (True / False) ________________       |
                                             |
(1.16) (True / False) ________________       |
                                             |
(1.17) (True / False) ________________       |
                                             |
(1.18) _________________________ bytes       |
                                             |
```