Final Exam SECTION 001 Dec 21, 2017

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

About this exam:

- There are 2 problems totaling 100 points:

  Problem 1: 54 points
  Problem 2: 46 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.

```
+----------------------------------------------------------------------+
|            This exam is for SECTION 001, 11:40am class.              |
|      You MUST be registered for this section to take this exam.      |
+----------------------------------------------------------------------+
|  PLEASE DO NOT OPEN THIS EXAM BOOKLET UNTIL YOU ARE TOLD TO DO SO!  |
+----------------------------------------------------------------------+
References: SmartPtr, classify_pointer(), print(), PRINT_TYPE()
----------------------------------------------------------------
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) {
                if (--*count == 0) { delete count; delete ptr; }
                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; }
};

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; }

/* Assume that PRINT_TYPE( <expression> ) will print the type name of
 * the given expression. For example,
 *
 *   int x;
 *   PRINT_TYPE( &x );
 *
 * will print the following:
 *
 *   int*
 *
 * PRINT_TYPE() is implemented as a macro and the code is not shown here.
 */
```

Problem [1] (18 parts, 54 points total) Consider the following C++ program:

```
struct Node {
    const char *data;
    SmartPtr<Node> next;

    Node(const char *d) : data(d), next(0) {}
    Node(const char *d, const SmartPtr<Node>& n) : data(d), next(n) {}
};

MyString f(SmartPtr<Node> node) {
    if (node)
        return f(node->next) + node->data;
    else
        return "";
}

int main()
{
    SmartPtr<Node> a(new Node("A"));
    SmartPtr<Node> b(new Node("B", a));
    SmartPtr<Node> c(new Node("C", b));

    assert(’B’ == 66);

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

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

    cout << "\n (1.3) "; classify_pointer( b->data );

    cout << "\n (1.4) "; classify_pointer( &(b->data) );

    cout << "\n (1.5) "; classify_pointer( b->next );

    cout << "\n (1.6) "; classify_pointer( &(b->next) );

    cout << "\n (1.7) "; print((int)( (void *)(b->next->next) == NULL ));

    cout << "\n (1.8) "; print((int)(        *((*b).data + 1)         ));

    cout << "\n (1.9) "; print( b->next->data );

    cout << "\n(1.10) "; print( f(c) );

    SmartPtr<Node> *a2 = new SmartPtr<Node>(a);

    cout << "\n(1.11) "; PRINT_TYPE( &**a2 );

    cout << "\n(1.12) "; print( ref_count(a) );

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

    cout << "\n(1.14) "; print( ref_count(c) );

    /* (1.15) - (1.18): Questions on memory leak */
}
```

Problem [1] continued

(1.1) - (1.14)

The program builds without error, runs without crashing, and successfully produces 14 lines of output for (1.1) ... (1.14). Fill in the blanks on the answer sheet to match the program’s output.

(1.15)

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.)

(1.16)

How many Node objects are leaked at the end of the program? Write the number of objects, not the number of bytes. (Write 0 for no leak.)

(1.17)

How many SmartPtr<Node> objects are leaked at the end of the program? Write the number of objects, not the number of bytes. (Write 0 for no leak.)

Do NOT count the SmartPtr<Node> objects that are embedded in a Node object (i.e., the ’next’ members of Node objects).

(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] (46 points total)

Consider the following definition of struct Pt:

```
        struct Pt {
            int x;
            int y;

            Pt()                { x = 1; y = 2; }
            void swap()         { int t = x; x = y; y = t; }
            void print() const { cout << x << y << endl; }
        };
```

Given the definition of Pt, determine the output of each of the twelve programs, (2.1) - (2.12). All twelve programs build successfully and all assert() calls succeed. Here is what you write on the answer sheet:

- Write "BAD" if Valgrind reports at least one memory error that is not a memory leak. In addition, there may or may not be memory leaks.

- Write "LEAK" if Valgrind reports that the program has memory leak, but no other types of memory error are reported.

- If there are no memory leaks or other memory errors:

    1) Write the actual output of the program, or

    2) Simply write "UNPREDICTABLE" if the output depends on undefined behaviors or the output can vary from one run to another.

If you write BAD, LEAK, or UNPREDICTABLE, write nothing else. If you write the output of the program in addition to BAD, LEAK, or UNPREDICTABLE, your answer will be considered a shotgun answer and will receive no credit.

(2.1)

```
        void transpose(Pt *p) { p->swap(); }
        int main()            { Pt p; transpose(&p); p.print(); }
```

(2.2)

```
        void transpose(Pt& p) { p.swap(); }
        int main()            { Pt p; transpose(p); p.print(); }
```

(2.3)

```
        void transpose(Pt p)  { p.swap(); }
        int main()            { Pt p; transpose(p); p.print(); }
```

(2.4)

```
        int main() { Pt p; cout << (p.x | p.y) << (p.x << p.y) << endl; }
```

(2.5)

```
SmartPtr<Pt> transpose(Pt p) {
    p.swap();
    SmartPtr<Pt> sp(new Pt(p));
    return sp;
}
int main() {
    Pt p;
    SmartPtr<Pt> sp = transpose(p);
    sp->print();
}
```

(2.6)

```
int main() {
    Pt *p = new Pt();
    SmartPtr<Pt> sp1(p); sp1->swap();
    SmartPtr<Pt> sp2(p); sp2->swap();
    cout << sp1->x << sp1->y
         << sp2->x << sp2->y << endl;
}
```

(2.7)

```
Pt& operator+(Pt p1, Pt p2) {
    Pt p3;
    p3.x = p1.x + p2.x;
    p3.y = p1.y + p2.y;
    return p3;
}
int main() {
    Pt p1, p2;
    Pt p3 = p1 + p2;
    p3.print();
}
```

(2.8)

```
int main() {
    assert(sizeof(Pt) == sizeof(Pt*));
    Pt p;
    vector<Pt> v;
    v.push_back(p);
    for (int i = 0; i < 10; i++) { v.push_back(p); }
    assert(v.size() == 11);
    cout << (char *)&v[10] - (char *)&v[0] << endl;
}
```

(2.9)

```
int main() {
    Pt p;
    vector<Pt> v;
    v.push_back(p);
    Pt *p0 = &v[0];
    for (int i = 0; i < 20; i++) { v.push_back(p); }
    assert(v.size() == 21);
    cout << (char *)&v[20] - (char *)p0 << endl;
}
```

(2.10)

```
int main() {
    Pt *p = new Pt();
    if (fork() == 0) { // child process
        p->print();
        return 0;
    }
    // parent process
    p->print();
    delete p;
    return 0;
}
```

(2.11)

```
static Pt p;

int main(int argc, char **argv) {
    if (argc == 1) { // no command line argument
        assert(p.x == 1 && p.y == 2);
        p.swap();
        execl("./a.out", "a.out", "hello,", (char *)0);
        cout << "bye" << endl;
    } else {
        cout << argv[1];
        p.print();
    }
}
```

We ran the program with no command line arguments:

```
./a.out
```

(2.12)

```
int main() {
    FILE *f = fopen("tempfile", "w");
    Pt p, p_net;
    p_net.x = htonl(p.x); // host-to-network-long
    p_net.y = htonl(p.y);
    fwrite(&p_net, sizeof(Pt), 1, f);
    fclose(f);

    f = fopen("tempfile", "r");
    unsigned char c;
    while (fread(&c, 1, 1, f)) printf("%d,", (int)c);
    printf("\n");
    fclose(f);
}
```

[blank page]

COMS 3157, Fall 2017: Final Exam, SECTION 001 Answer Sheet,   page 1 of 2

UNI: Name:

----

[2] Please read the problem description carefully. In summary, do ONE of the following:

- Write "BAD" and nothing else.
- Write "LEAK" and nothing else.
- Write "UNPREDICTABLE" and nothing else.
- Write the output of the program.

(2.1) ______________________________

(2.2) ______________________________

(2.3) ______________________________

(2.4) ______________________________

(2.5) ______________________________

(2.6) ______________________________

(2.7) ______________________________

(2.8) ______________________________

(2.9) ______________________________

(2.10) ______________________________

(2.11) ______________________________

(2.12) ______________________________

```
COMS 3157, Fall 2017                  +---+---+---+---+---+---+---+---+
Final Exam, SECTION 001     Your UNI: |   |   |   |   |   |   |   |   |
Answer Sheet, page 2 of 2             +---+---+---+---+---+---+---+---+
```

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) __________ MyString objects

(1.16) __________ Node objects

(1.17) __________ SmartPtr<Node> objects

(1.18) __________ bytes