Final Exam
Dec 20, 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: 24 points
  problem 2: 24 points
  problem 3: 30 points
  problem 4: 22 points

- Assume the following programming environment:

  Platform: Ubuntu Linux 12.04, 64-bit version
  Primitive type sizes: 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.

What to hand in and what to keep:

- At the end of the exam, you will hand in only the answer sheet, which contains the last 2 pages (a single sheet printed double-sided.)

- 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!

Problem [1]: 24 points

Consider the following CLIC lab session:

```
    ~/cs3157$ git clone /home/jae/cs3157-pub/lab1 lab1
    Initialized empty Git repository in ~/cs3157/lab1/.git/

    ~/cs3157$ cd lab1

    ~/cs3157/lab1$ ls
    main.c  Makefile  myadd.c  myadd.h  README.txt

    ~/cs3157/lab1$ git status
    # On branch master
    nothing to commit (working directory clean)

    ~/cs3157/lab1$ vim  Makefile  myadd.h  multiply.c

    ~/cs3157/lab1$ make
    gcc -g -Wall    -c -o main.o main.c
    gcc -g -Wall    -c -o myadd.o myadd.c
    gcc -g  main.o myadd.o   -o main

    ~/cs3157/lab1$ make
    make: ‘main’ is up to date.

    ~/cs3157/lab1$ touch myadd.h

    ~/cs3157/lab1$ make
    gcc -g -Wall    -c -o main.o main.c
    gcc -g -Wall    -c -o myadd.o myadd.c
    gcc -g  main.o myadd.o   -o main

    ~/cs3157/lab1$ ls
    main    main.o    multiply.c  myadd.h  README.txt
    main.c  Makefile  myadd.c     myadd.o

    ~/cs3157/lab1$ git add myadd.h

    ~/cs3157/lab1$
```

Note that the user issued "vim Makefile myadd.h multiply.c" command to launch his editor, modified some comments in Makefile and myadd.h, and created a new file, multiply.c, to start working on a multiplication routine.

Problem [1] continued

(a) 18 points

At the end of the shell session shown above, there are 9 files in the current directory:

```
~/cs3157/lab1$ ls
main    main.o    multiply.c  myadd.h  README.txt
main.c  Makefile  myadd.c     myadd.o
```

Put each file into one of the following 4 categories from the point of view of the git revision control system.

Untracked:
Tracked, unmodified:
Tracked, modified, but unstaged:
Tracked, modified, and staged:

(b) 6 points

The Makefile in that directory is shown here, with the build rules removed. Fill in the missing lines, specifying targets and dependencies that are consistent with the CLIC lab session shown above.

Fill in only targets and dependencies. That is, do not write the gcc commands that come below the target/dependency lines. Rely on the implicit rules that make knows by default.

```
CC  = gcc
CXX = g++
INCLUDES =
CFLAGS   = -g -Wall $(INCLUDES)
CXXFLAGS = -g -Wall $(INCLUDES)
LDFLAGS = -g
LDLIBS =

# Fill in the missing targets and dependencies here

.PHONY: clean
clean:
        rm -f *.o *~ a.out core main
```

Problem [2]: 24 points

Multiple-choice questions: please write down exactly ONE choice (from a,b,c,d,e) clearly on the answer sheet provided.

(2.1) What is the output of the following program?

```
#include <stdio.h>
#include <unistd.h>

static int sum = 0;

int main()
{
    int i;
    for (i = 0; i < 5; i++) {
        if (fork() == 0) {
            // child process
            sum++;
            return sum;
        }
    }
    printf("%d\n", sum);
    return 0;
}
```

(a) 0
(b) 5
(c) Nothing gets printed.
(d) It varies between 0 and 5 depending on how the operating system schedules the parent and child processes.
(e) None of the above.

(2.2) Which of the following is NOT a correct statement about using the -fno-elide-constructors compiler flag that we used in lab 9?

(a) It prevents certain types of memory leaks.
(b) It disables certain optimizations.
(c) It may increase the number of copy constructor invocations.
(d) It may increase the number of temporary objects created.

(2.3) Which of the following is a correct description of the fdopen() library function?

(a) It is the debugging version of fopen().
(b) It returns a file descriptor, which is of type int.
(c) It can only be used for reading from a file.
(d) It provides a wrapper on top of a UNIX-style file descriptor.

Problem [2]: continued

(2.4) Which Internet protocol layer is responsible for providing reliable byte stream where data arrives in-order in a virtual pipeline?

(a) The link layer (commonly Ethernet).
(b) The internet layer (IP).
(c) The transport layer (TCP).
(d) The application layer (for example HTTP).
(e) None of the above.

(2.5) What do we call the process of creating an executable program from a number of object files?

(a) Preprocessing
(b) Compiling
(c) Linking
(d) None of the above.

(2.6) The prototypes of some of the member & friend functions of MyString class are listed below. Which of them is NOT correct?

(a) ostream& operator<<(ostream& os, const MyString& s)
(b) MyString& MyString::operator=(const MyString& rhs)
(c) MyString& MyString::operator+=(const MyString& s)
(d) MyString& operator+(const MyString& s1, const MyString& s2)
(e) All of the prototypes above are correct.

(2.7) Which of the following is a correct statement about the execution of the following program?

```
int main()
{
    SmartPtr<MyString> sp(new MyString("hello"));
    vector<SmartPtr<MyString> >  v;
    v.push_back(sp);
    cout << *v[0] << endl;
    return 0;
}
```

(a) The program can crash.
(b) The program has a memory leak.
(c) The smart pointer's reference count (i.e., *sp.count) was never more than 1.
(d) None of the statements above are correct.

(2.8) Which of the following declarations is NOT a declaration for an array?

(a) int foo(int a[10]);
(b) int (*f3[5])(void *v1, void *v2);
(c) char (*(*x[3])())[5];
(d) int *daytab[13];

Problem [3]: 30 points (10 expressions, 3 points each)

Evaluate the 10 expressions in the context of the given 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.

- Treat boolean expressions as integer expressions. Please do not write "true" or "false". Write 1 for true and 0 for false.

- 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:

- 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 [3]: continued

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

```
    int main(int argc, char **argv)
    {
        // settings

        MyString s[3];
        s[0] = "0";
        s[1] = "1";
        s[2] = "2";
        vector<MyString>   v;
        vector<MyString*>  vp;

        char c[2] = { 0, 0 };
        for (int i = 0; i < 3; i++) {
            *c = '0' + i;

            v.push_back( c );
            vp.push_back( &s[i] );
        }
```

Evaluate the following 10 expressions:

(3.1) `s[2] == v[2]`

(3.2) `&s[2] == vp[2]`

(3.3) `v[2] == *vp[2]`

(3.4) `&v[2] == vp[2]`

(3.5) `s[0][0] == 0`

(3.6) `strlen( &v[0][0] + 1 )`

(3.7) `vp[2]`

(3.8) `v.size() <= v.capacity()`

(3.9) `sizeof(c)`

(3.10) `v.end() - v.begin()`

Problem [4]: 22 points

Consider malgrind.c that defines MALLOC() and FREE():

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

// 64-bit unsigned integer with all bits set to 1
static uint64_t MAGIC = ~((uint64_t)0);

const size_t WAD = sizeof(MAGIC);

void *MALLOC(size_t n)
{
    char *p = (char *) malloc(2 * WAD + n);
    if (p == NULL)
        return NULL;

    uint32_t *head = (uint32_t *)p;
    head[0] = head[1] = n;
    memcpy(p + WAD + n, &MAGIC, WAD);
    return p + WAD;
}

void FREE(void *p)
{
    if (p) {
        uint32_t *head = (uint32_t *)p - 2;
        if (head[0] != head[1] ||
                memcmp((char *)p + head[0], &MAGIC, WAD) != 0) {
            fprintf(stderr, "%s\n", "OOPS");
        } else {
            fprintf(stderr, "%s\n", "OKAY");
        }

        free( (char *)p - WAD );
    }
}
```

Here are some excerpts from memcmp & memcpy man pages for you to reference:

```
int memcmp(const void *s1, const void *s2, size_t n);
```

  The memcmp() function compares the first n bytes of the memory areas s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

```
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 [4]: continued

Here is a test program mgtest1.c, which uses the MALLOC & FREE functions defined in malgrind.c:

```
// mgtest1.c

#include <stdio.h>
void *MALLOC(size_t n);
void FREE(void *p);

int main()
{
    int *a = (int *) MALLOC(4 * sizeof(int));
    int i;
    for (i = 0; i <= 4; i++) {
        a[i] = i;

        // print the integer values
        fprintf(stderr, "%d, ", a[i]);
    }

    FREE(a); // FREE() will print OOPS or OKAY
    return 0;
}
```

(4.1) 3 points

We compiled, linked, and valgrind-tested the program as follows:

```
$ gcc -g -Wall mgtest1.c malgrind.c
$ valgrind --leak-check=yes ./a.out
```

Which of the following correctly describes the result of valgrind-testing?

CRASH: Valgrind detects memory errors that can make the program crash.
LEAK: Valgrind reports memory leaks, but no other memory errors.
GOOD: Valgrind reports no memory leak and no memory error.

Please write one of "CRASH", "LEAK", or "GOOD" on your answer sheet.

(4.2) 6 points

We compiled, linked, and ran the program as follows:

```
$ gcc -g -Wall mgtest1.c malgrind.c
$ ./a.out
```

Write the output of the program. If you think the program may crash, write the most likely output if the program happens to run without crashing.

Problem [4]: continued

We now replace mgtest1.c with a slightly modified mgtest2.c:

```
// mgtest2.c

#include <stdio.h>
void *MALLOC(size_t n);
void FREE(void *p);

int main()
{
    int *a = (int *) MALLOC(4 * sizeof(int));
    int i;
    for (i = 0; i <= 4; i++) {
        a[i] = i;
    }

    int *p;
    for (p = a - 2; p < a + 6; p++) {
        fprintf(stderr, "%d, ", *p);
    }

    FREE(a); // FREE() will print OOPS or OKAY
    return 0;
}
```

(4.3) 3 points

Same thing as (4.1), but with mgtest2.c instead.

Please write one of "CRASH", "LEAK", or "GOOD" on your answer sheet.

(4.4) 6 points

Same thing as (4.2), but with mgtest2.c instead.

Write the output. (Again, the most likely output if the program can crash.)

(4.5) 4 points

Which of the following best describes the potential benefit of using the MALLOC() & FREE() wrappers instead of using malloc() & free() directly?

(a) Using MALLOC/FREE can make the program run faster.
(b) Using MALLOC/FREE can help detect memory leaks.
(c) Using MALLOC/FREE can help detect writing to invalid locations.
(d) All of the above.

Please write exactly one of a, b, c, or d on your answer sheet.

COMS W3157, Fall 2012: Final Exam Answer Sheet, Page 1 of 2

Name: _____

UNI: _____

[1]

(a)

Untracked:

Tracked, unmodified:

Tracked, modified, but unstaged:

Tracked, modified, and staged:

(b) # Fill in the missing targets and dependencies

---

[2]

(2.1) _____

(2.2) _____

(2.3) _____

(2.4) _____

(2.5) _____

(2.6) _____

(2.7) _____

(2.8) _____

[3]

(3.1) s[2] == v[2]

(3.2) &s[2] == vp[2]

(3.3) v[2] == *vp[2]

(3.4) &v[2] == vp[2]

(3.5) s[0][0] == 0

(3.6) strlen(&v[0][0] + 1)

(3.7) vp[2]

(3.8) v.size() <= v.capacity()

(3.9) sizeof(c)

(3.10) v.end() - v.begin()

[4]

(4.1) mgtest1 (CRASH,LEAK,GOOD): _____

(4.2) mgtest1 output:

(4.3) mgtest2 (CRASH,LEAK,GOOD): _____

(4.4) mgtest2 output:

(4.5) Choose one of a, b, c, d: _____