Midterm Examination

COS 217, Fall 2004

Your name: _____

Your NetID: _____

Your preceptor: _____

There are 5 problems on this examination, worth a total of 100 points. You have 50 minutes to complete this examination.

You are not required to comment your code in this exam unless what you have written is difficult to understand.

This examination is open book and open notes, but no calculators or other electronic devices are permitted.

For some of the problems, partial credit will be given if you can show your work in arriving at solutions. But ***be brief***.

Good luck!

Write out and sign the Honor Code pledge before turning in the test.

"I pledge my honor that I have not violated the Honor Code during this examination.

Problem	Worth	Earned
1	14	
2	16	
3	15	
4	20	
5	35	
Total	100	

1. Multiple choice questions. (14 points)

For each of the following questions, enumerate ***all*** applicable answers. (No explanation is necessary.)

(a)

```
1       int i = 0;
2       int *p = &i;
3       /* insert printf() or scanf() statement here */
```

Suppose we insert one of the following statements on line 3, and compile the resulting program (ignoring any compile time ***warnings***, if any). Which of these lines will result in either a fatal compile time error or a fatal run time error?

A. `printf("%d", &i);`

B. `printf("%d", *i);`

C. `printf("%d", p);`

D. `printf("%d", &p);`

E. `printf("%d", *p);`

F. `scanf("%d", i);`

G. `scanf("%d", p);`

H. `scanf("%d", &p);`

I. `scanf("%d", *p);`

(b) Which of the following type of files are guaranteed to continue to work when they are copied from a Sun machine to an Intel PC?

A. `*.java`
B. `*.class`
C. `*.c`
D. `*.s`
E. `*.o`
F. `a.out`

(c) Which of the following statements are true?

A. It is a good idea to remove all the `assert()` statements (by either deleting them or commenting them out in an editor) when we are ready to release "production-quality" software to customers, so that these `assert()` statements do not unnecessarily slow down the released program.
B. It is a good practice to profile and optimize code as you write it, so you can isolate performance problems early on.
C. A "stack trace" allows you to examine the values of the local variables in the active procedure calls at the time the program is stopped or terminated.
D. If you modify only a single `.c` file, it is always the case that none of the other `.c` files that contribute to the same program would need to be recompiled.

(d) If a user provides wrong input, which ones of the following are acceptable behaviors of a "working" program.

A. Prints an error message and calls `exit()`.
B. Prints an error message and calls `abort()`.
C. Triggers an `assert()` statement, which provides a meaningful feedback on what's wrong, and then asks the user to try again.
D. Provides a core dump so the user can find out which parts of the code are responsible for rejecting the input.

2. Scoping. (16 points)

Give the output of the following program (which is broken down into two files: `f1.c` and `f2.c`).

`f1.c`

```
#include <stdio.h>

int x;
static int y;

void f() {
    x = 2;
    y = 3;
}

void g(int a) {
    int x = 1;
    static int y;

    if (a == 0) {
        y = 0;
    } else {
        y += x;
    }
    printf("g: %d %d\n",
           x, y);
}

void p() {
    printf("p: %d %d\n",
           x, y);
}
```

`f2.c`

```
#include <stdio.h>

extern int x;
static int y;

void s() {
    printf("s: %d %d\n",
           x, y);
}

main() {
    g(0);
    f();
    g(1);
    p();
    s();
}
```

3. Arrays and pointers. (15 points)

```
#include <stdio.h>

void f(void *p[]) {
    void *q;
    q = *p;
    *p = *(p+1);
    *(p+1) = q;
}

main(int argc, char **argv) {
    f((void **)argv);
    printf("%s\n", argv[0]);
}
```

Suppose the above file is named `f.c`, and we type the following commands into a Unix shell:

```
% gcc f.c
% a.out 23 45
```

(a) Say with a single English sentence what function `f()` does.

(b) What's the output produced by this program?

4. Dynamic memory. (20 points)

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04
05  #define MAX 100
06
07  static char *read_next_line(FILE *fp) {
08      char *p;
09
10      p = (char *) malloc(MAX);
11      if (p == NULL)
12          return NULL;
13
14      p = fgets(p, MAX, fp);
15      /* char *fgets(char *s, int size, FILE *stream);
16         fgets() reads in at most one less than size characters from
17         stream and stores them into the buffer pointed to by s.
18         Reading stops after an EOF or a newline. A '\0' is stored
19         after the last character in the buffer.  fgets() return s on
20         success, and NULL on error or when end of file occurs while no
31         characters have been read. */
32      return p;
33  }
34
35  static char *find_max(char *s, char *t) {
36      char *max;
37
38      if (strcmp(s, t) >= 0)
39          max = s;
40      else
41          max = t;
42
43      return max;
44  }
45
```

(Code continues on the next page.)

```
46  void print_max_string_from_file (FILE *fp) {
47      char *p, *q, *r, *max1, *max2;
48
49      p = read_next_line(fp);
50      q = read_next_line(fp);
51      r = read_next_line(fp);
52
53      if (p == NULL || q == NULL || r == NULL)
54          return;
55
56      max1 = find_max(p, q);
57      free(p);
58      free(q);
59
60      max2 = find_max(max1, r);
61      free(max1);
62      free(r);
63
64      printf("The max of the 3 strings is <%s>\n", max2);
65
66      free (max2);
67  }
```

(a) Explain ***succinctly*** two ways that calling `print_max_string_from_file()` can leak memory. (Give line numbers of the leaks.)

(b) Explain ***succinctly*** two ways dangling pointers can be referenced when `print_max_string_from_file()` is called. (Give line numbers of the dangling references.)

5. ADTs. (35 points)

You're given the following interface of a symbol table ADT in a **symtable.h** file. (This is the same interface as the one discussed in lectures.)

```
typedef struct SymTable *SymTable_T;

/* create a new, empty table. */
SymTable_T SymTable_new(void);

/* enter (key,value) binding in the table; else return 0 if already there */
int SymTable_put(SymTable_T table, char *key, void *value);

/* look up key in the table, return value (if present) or else NULL */
void *SymTable_get(SymTable_T table, char *key);

/* apply f() to every binding in the table ... */
void SymTable_map(SymTable_T table,
   void (*f)(char *key, void *value, void *extra),
   void *extra);

/* Return the number of bindings in table.
It is a checked runtime error for table to be NULL. */
int SymTable_getLength(SymTable_T table);

/* Remove from table the binding whose key is key. Return 1 if successful, 0 otherwise.
  It is a checked runtime error for table or key to be NULL. */
int SymTable_remove(SymTable_T table, char *key);
```

You are to add a new function to the symbol table interface:

```
SymTable_T SymTable_copy(SymTable_T table, ...);
```

This function makes a copy of the input symbol table and returns this new copy. The two resulting symbol tables should be completely independent of each other, so for example, the client could free all the memory associated in any way with the first table, or alter the values stored in the first table, without affecting the second table in any way. (This is called a "deep copy.")

(a) Give the complete prototype (interface) of the `SymTable_copy` function. (Hint: you may need extra arguments in addition to the old symbol table.)

(b) Give an implementation of the `SymTable_copy` function. Note that you may ***not*** assume knowledge of how the existing symbol table ADT is implemented (in terms of, for example, whether it is implemented as a linked list or a hash table). (A correct answer could take less than 20 lines of code, although we won't penalize you if you use a few more.)