Princeton University
COS 217: Introduction to Programming Systems
Fall 2002
Midterm Exam Answers

Question 1

(1) `int *va[4];`
(2) `int (*vb[3])[4];`
(3) `void (*vc)(void);`
(4) `char (*vd[2])(int);`
(5) `char *(*(*ve)(char*))(int[]);`

Question 2

(1) a is an array of 5 pointers to float
(2) b is a pointer to an array of 2 arrays of 3 chars
(3) f is a pointer to a function that accepts a char and an int and returns a pointer to int
(4) f is a function that accepts a pointer to int and a pointer to an array of floats and returns a pointer to char
(5) g is a function that accepts an int and returns a pointer to a function that accepts an int and returns a char

Question 3

Here is the corrected program, with comments marking the corrections:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLESIZE 10000

/* (1) Must use call by reference */
void swap(char *a, char *b) {
  char t = *a;
  *a = *b; *b = t;
}

char *sort(char *s) {
  char *p; int i,j;
  int n = strlen(s);

  /* (2) Must allocate memory for the p string. */
  p = (char*)malloc(n + 1);

  /* (3) Must copy n+1 characters (thus including the '\0' character)
     to p.  Or could call the strcpy function. */
  for (i=0; i<=n; i++)
    p[i]=s[i];

  for (i=0; i<n; i++)    /* no bugs in this line */
```

```
    for (j=0; j<i; j++)  /* no bugs in this line */
      if (p[j]>p[j+1])   /* no bugs in this line */

        /* (1 continued) Must pass addresses to swap. */
        swap(&p[j], &p[j+1]);

  return p;
}

struct entry {char *key, *value;};
struct entry table[TABLESIZE];
static int numentries=0;

void addToTable(char *key, char *value) {

  /* (5) Must store a copy of value. */
  char *valuecopy;

  /* (4) Must avoid overflowing table. */
  if (numentries >= TABLESIZE)
     return;

  table[numentries].key=key;

  /* (5 continued) Must insert a *copy of* value into the table. */
  valuecopy = (char*)malloc(strlen(value) + 1);
  strcpy(valuecopy, value);
  table[numentries].value=valuecopy;

  numentries++;
}

/* Find all pairs of entries in the table that sort to the same thing. */
void printAnagrams() {
  int i,j;
  for(i=0; i<numentries; i++)
    for(j=0; j<i; j++)

      /* (6) Must compare strings, not pointers. */
      if (strcmp(table[i].key, table[j].key) == 0)

        printf("%s is an anagram of %s\n",
          table[i].value, table[j].value);
}

int main (int argc, char **argv) {
  char buf[100];

  /* (7) The gets function might overflow buf.  Should use fgets instead. */
  while (fgets(buf, 100, stdin)) {

    addToTable(sort(buf), buf);
  }
  printAnagrams();
  /* don't worry about calls to free() for this exam problem */
  return 0;
}
```

Question 4

```
/* tictac.c */

#include <stdio.h>
#include <stdlib.h>
#include "tictac.h"

/* data structure for representation of a Board_T */

struct board {
  player square[3][3];
  player whoseTurn;
};

/* Function to tell whose turn it is. */

player Board_whoseTurnIsIt(Board_T b) {

  /* Assuming that Board_new, Board_makeMove, and Board_unmakeMove
     maintain the whoseTurn field... */

  return b->whoseTurn;
}

/* Function to apply a function to the contents of every square */

void Board_foreach(Board_T b,
                   void (*f)(int row, int col, player p, void *extra),
                   void *extra) {
  int row;
  int col;
  for (row = 0; row < 3; row++)
    for (col = 0; col < 3; col++)
      (*f)(row, col, b->square[row][col], extra);
}


/* client.c */

#include <stdio.h>
#include <stdlib.h>
#include "tictac.h"

void incrementCount(int row, int col, player p, void *extra) {
  int *count = (int*)extra;
  if (p == X)
    (*count)++;
}

int countTheXes(Board_T b) {
  int count = 0;
  Board_foreach(b, incrementCount, &count);
  return count;
}
```

Question 5

```
10 100 1000
10 0 1000 10000 100000
11 100 1000
12 1 1001 10000 100001
13 101 1001
```

Explanation:

The “a” defined in one.c has file scope, external linkage, and process duration. The “a” declared in two.c also has file scope, external linkage, and process duration. The two “a” variables share the same storage; changing one affects the other.

The “b” defined in one.c has file scope, **external** linkage, and process duration. The “b” defined in two.c has file scope, **internal** linkage, and process duration. The two “b” variables do not share the same storage; changing one does not affect the other. As with all variables that have process duration, the “b” defined in two.c is initialized (to 0, by default) upon program startup.

The “c” defined in one.c has file scope, external linkage, and process duration. The “c” defined in two.c (as a function formal parameter) has block scope, internal linkage, and temporary duration. The two “c” variables occupy different storage locations; changing one does not affect the other. The “c” in two.c is created upon entrance to the f function, and is destroyed upon return from that function.

The “d” defined in two.c has block scope, internal linkage, and temporary duration. It is created and initialized upon entrance to the f function, and is destroyed upon return from that function.

The “e” defined in two.c has block scope, internal linkage, and **process** duration. It is created and initialized at program startup. It continues to exist after return from the f function.

Question 6

Error! Not a valid link.
Alternate answer:

Error! Not a valid link.

Question 7

The answer is team specific.

Some general observations:

- In the .h files, the Delta, Move, and GameState data structures would need to be modified. All function declarations whose parameters are specific to those data structures would need to be modified. All comments that are specific to those data structures or to the game of Kalah would need to be modified.
- The .c file that contains the main function would not need to be modified. The .c file that contains the alpha beta search function(s) would not need to be modified. All other .c files would need to be modified.

Some notable exceptions:

- In principle it might be possible to use the Delta, Move, and/or GameState data structures without modification to implement other games. If you explained how, you received full credit.
- The Red team's main.c file reads and writes Moves, and so would need to be modified.
- The Orange team's Kalah.c file reads and writes Moves, and so would need to be modified.
- The Orange team's AlphaBeta.c file uses an array of length 6 to store moves. The length of that array would need to be modified. That file also assumes that each Move is implemented as an int, and so would need to be modified.
- The Gold team defines Move and Delta as ADTs via opaque pointers. Thereby the Gold team's .h files would require fewer modifications than those of other teams.