Exercise 0 *(6 points – 1 point per question – No program required)*

1. B
2. D
3. C
4. C
5. D
6. C

References and Explanations:
In addition to the course book references cited below, these topics are also covered in the live lectures (in-class students) and the recorded lectures (online students).

1. Notes 1.11, 6.4, 8.1; The first and only required argument of *printf* is called the "control string" (or the "format string"). Any characters in the control string other than those interpreted as "conversion specifications" are printed literally. A conversion specification is most often used to cause the value of a subsequent *printf* argument to be output in its place. Most commonly a string literal is used as the first argument of *printf*, but this is not required. Since a string literal is an array of constant characters and since all arrays decay to a pointer to their first element unless used as the sole operand of the address operator or the **sizeof** operator, a string literal will decay to a character pointer when used as a function argument. Thus, what is really being passed as first argument of *printf* is a character pointer. It doesn't matter whether that character pointer resulted from the decay of a character array or was originally declared to be a character pointer. All that matters is that the string of characters it points to eventually ends with a null character.

2. Note 9.9; Unlike arrays, structures do not decay under any circumstances. Thus, when a structure is passed to or returned from a function argument what gets passed or returned is a copy of it. Since copying large objects can be very expensive in terms of system overhead, it should be avoided if possible and pointers or possibly references (C++ only) to them should be passed or returned instead.

3. Note 8.4; Dynamic memory allocations must always be tested for success/failure.

4. Note 8.4; Dynamic memory allocations are freed by calling *free* (in C) or ***delete*** (in C++) and providing the address that was obtained when the memory was allocated. In this quiz question the value of that address was lost when variable *vp* was overwritten in the second statement.

5. Note 9.10; For answer D the Right-Left rule describes parameter *junk* as "…a pointer to **struct** junk". A pointer to a structure is formed (like pointers to other types) by placing the address operator to the left of the expression representing that structure.

6. Notes 6.16, 7.3, 8.1, 8.2; The *printf* control string contains three space-separated %s conversion specifications. In *printf* the %s conversion specification requires its corresponding argument to be a character pointer and will print a string of characters starting at that address until a null character, '\0', is reached. In order to print the output string required by this quiz question, three character pointers are required that point to the appropriate characters in the string literals whose pointers are stored in array *p*. Specifically, the first pointer must point to the *'t'* in *"now's the"*, the second pointer must point to the *'b'* in *"…brown"*, and the third pointer must point to the *'l'* in *"my letter…"*. In the various answer choices, the three arguments point to the following, respectively:

    A. 't' in "now's the", -- out of bounds; not a char pointer --, 'l' in "my letter…"
    B. 't' in "…the mail", 'l' in "my letter…", 'l' in "my letter…"
    C. 't' in "now's the", 'b' in "…brown" , 'l' in "my letter…"
    D. -- out of bounds; not a char pointer --, 'b' in "…brown" , 'l' in "my letter…"

Exercise 1 *(7 points – C++ Program)*

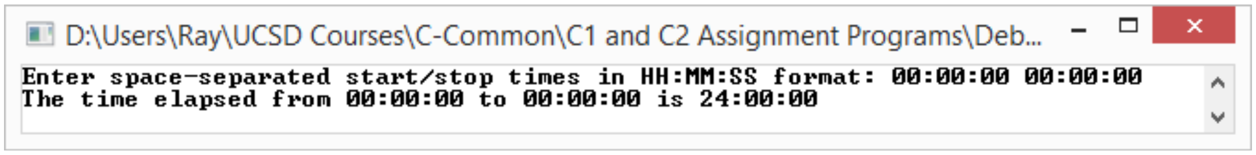
```
   ***************************************** FILE C1A7E1_MyTime.h *****************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains the definition of structure type MyTime
// and a prototype for the DetermineElapsedTime function.
//

#ifndef C1A7E1_MYTIME_H
#define C1A7E1_MYTIME_H

// Define structure type to represent a time.
struct MyTime { int hours, minutes, seconds; };

MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop);

#endif


   ******************************** FILE C1A7E1_DetermineElapsedTime.cpp ********************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function DetermineElapsedTime, which calculates
// and returns a pointer to the difference between the times pointed
// to by its parameters.
//

#include "C1A7E1_MyTime.h"

const long SEC_MN = 60;                 // seconds per minute
const long SEC_HR = 60 * SEC_MN;        // seconds per hour
const long SEC_DAY = 24 * SEC_HR;       // seconds per day, must be long

//
// Determine the amount of time elapsed between the times stored in the
// MyTime structures in <start> and <stop> (starting with the time in
// <start>) and store the result in MyTime structure <elapsed>.  If
// the time in <start> is greater than the time in <stop> the time
// in <stop> is considered to be in the next day.  Return a pointer
// to <elapsed>.
//
MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop)
{
   long startSec, stopSec, difference;
   static MyTime elapsed;

   // convert argument times into seconds
   startSec = start->hours * SEC_HR + start->minutes * SEC_MN + start->seconds;
   stopSec = stop->hours * SEC_HR + stop->minutes * SEC_MN + stop->seconds;

   difference = stopSec - startSec;                 // seconds elapsed
   if (difference <= 0)                             // time is in next day
      difference += SEC_DAY;                        // add day of seconds

   // convert difference back to hours, minutes, seconds
   elapsed.hours = int(difference / SEC_HR);       // hours
   difference %= SEC_HR;                           // seconds left
   elapsed.minutes = int(difference / SEC_MN);     // minutes
   difference %= SEC_MN;                           // seconds left
   elapsed.seconds = int(difference);              // seconds

   return(&elapsed);                               // return structure pointer
}
```

*** FILE C1A7E1_main.cpp ***

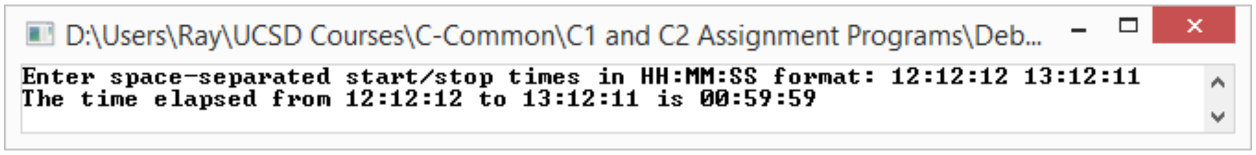
```
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function main, which prompts the user for pairs of
// military times and calls the DetermineElapsedTime function to determine
// the time difference.  That difference is displayed.
//

#include <iostream>
#include <iomanip>
#include <cstdlib>
using std::cin;
using std::cout;
using std::setfill;
using std::setw;

#include "C1A7E1_MyTime.h"

const int ITERATIONS = 3;        // how many tests to run

//
// Prompt the user for two times in military format and store them
// directly into the members of two MyTime structures.  Then call
// the DetermineElapsedTime function, passing pointers to those
// two structures as arguments.  Finally, display the elapsed time
// in the MyTime structure pointed to by the pointer returned by
// DetermineElapsedTime.  Do all of this ITERATIONS times.
//
int main()
{
   cout << setfill('0');
   for (int iterationCount = 0; iterationCount < ITERATIONS; ++iterationCount)
   {
      MyTime start, stop, *elapsed;
      char ch;

      // Get two times in military format from user.
      cout << "Enter space-separated start/stop times in HH:MM:SS format: ";
      cin >> start.hours >> ch >> start.minutes >> ch >> start.seconds
          >> stop.hours >> ch >> stop.minutes >> ch >> stop.seconds;

      // Determine the time difference between the two times.
      elapsed = DetermineElapsedTime(&start, &stop);
      cout << "The time elapsed from "
         << setw(2) << start.hours << ':'
```

```
            << setw(2) << start.minutes << ':'
            << setw(2) << start.seconds << " to "
            << setw(2) << stop.hours << ':'
            << setw(2) << stop.minutes << ':'
            << setw(2) << stop.seconds << " is "
            << setw(2) << elapsed->hours << ':'
            << setw(2) << elapsed->minutes << ':'
            << setw(2) << elapsed->seconds << '\n';
    }
    return EXIT_SUCCESS;
}
```

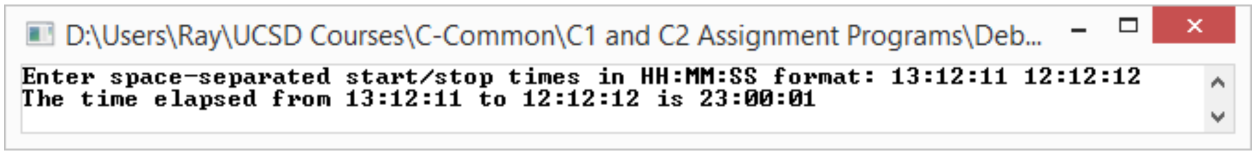
C1A7E1 Screen Shots


D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...

```
Enter space-separated start/stop times in HH:MM:SS format: 00:00:00 00:00:00
The time elapsed from 00:00:00 to 00:00:00 is 24:00:00
```


D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...

```
Enter space-separated start/stop times in HH:MM:SS format: 12:12:12 13:12:11
The time elapsed from 12:12:12 to 13:12:11 is 00:59:59
```


D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...

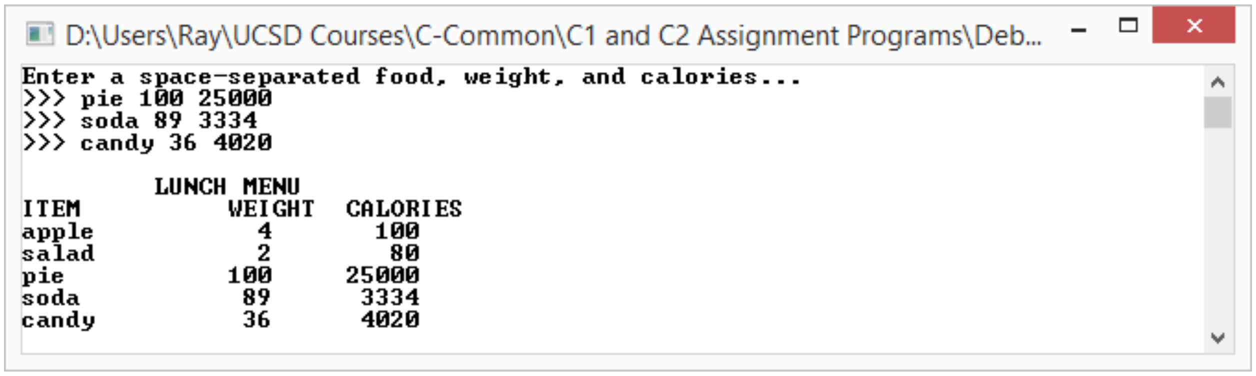
```
Enter space-separated start/stop times in HH:MM:SS format: 13:12:11 12:12:12
The time elapsed from 13:12:11 to 12:12:12 is 23:00:01
```

Exercise 2 (7 points – C Program)

```c
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function main, which prompts the user for information
 * about several foods, stores that information in memory, then displays a
 * table containing that information.
 */

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

#define LUNCH_ITEMS 5      /* total lunch menu items */
#define FIXED_ITEMS 2      /* items initialized when declared */
#define BUFSIZE 256        /* size of input buffer */
#define BUFFMT "%255"      /* scanf field width for buffer */

/*
 * Prompt the user to input information about some food items and store them
 * in structures in an array that already has some hard-coded food item
 * information stored in it.  The food name strings occupy only exactly the
 * amount of memory necessary to hold them (including the null terminator
 * character).  After the food item information has been input and stored
 * all food item information in the array is displayed and all dynamically-
 * allocated memory is freed.
 */
int main(void)
{
   int foodItemNo;

   /*
    * Define type struct Food, declare an array of struct Food objects, and
    * partially initialize the array.
    */
   struct Food
   {
      char *name;
      int weight, calories;
   } lunch[LUNCH_ITEMS] = {{"apple", 4, 100}, {"salad", 2, 80}};

   /*
    * Since members of the remaining structures have not been explicitly
    * initialized, each name member is a null pointer.  Each must, thus,
    * be initialized to point to an area of storage where the incoming
    * characters of the food name can be stored.
    */
   if (FIXED_ITEMS < LUNCH_ITEMS)
      printf("Enter a space-separated food, weight, and calories...\n");
   for (foodItemNo = FIXED_ITEMS; foodItemNo < LUNCH_ITEMS; ++foodItemNo)
   {
      size_t length;
      char buf[BUFSIZE];                   /* for getting name of food */
      printf(">>> ");
      scanf(BUFFMT "s %i %i", buf, &lunch[foodItemNo].weight,
         &lunch[foodItemNo].calories);
```

```
        length = strlen(buf) + 1;           /* characters used in buf + '\0' */

        /* allocate storage for the name pointer to point to */
        if ((lunch[foodItemNo].name = (char *)malloc(length)) == NULL)
        {
           fprintf(stderr, "malloc out of memory\n");
           exit(EXIT_FAILURE);
        }
        /* copy food into malloc buffer */
        memcpy(lunch[foodItemNo].name, buf, length);
     }

     printf("\n         LUNCH MENU\nITEM          WEIGHT  CALORIES\n");
     for (foodItemNo = 0; foodItemNo < LUNCH_ITEMS; foodItemNo++)
     {
        printf("%-13s%4i%10i\n", lunch[foodItemNo].name,
           lunch[foodItemNo].weight, lunch[foodItemNo].calories);
        if (foodItemNo >= FIXED_ITEMS)
           free(lunch[foodItemNo].name);
     }
     return EXIT_SUCCESS;
  }
```

C1A7E2 Screen Shot



D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...

```
Enter a space-separated food, weight, and calories...
>>> pie 100 25000
>>> soda 89 3334
>>> candy 36 4020

         LUNCH MENU
ITEM          WEIGHT  CALORIES
apple           4       100
salad           2        80
pie           100     25000
soda           89      3334
candy          36      4020
```