```
Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 146359, Ray Mitchell
June 25, 2019
C1A7E0_Quiz.txt
Quiz Answers

1. E
2. E
3. C
4. B
5. E
6. B
```

C1A7E0 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. **E** Note 9.9; Unlike arrays, structures and classes don't decay. Thus, when a structure or class is passed to or returned from a function what gets passed or returned is a copy of it. Since copying large objects can be very expensive in terms of system resources it should be avoided if possible and pointers (or references in C++ only) to them should be passed or returned instead.

2. **E** 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. That is exactly what is done in this quiz question.

3. **C** Notes 1.11, 6.4, 8.1; The first and only required argument of *printf* is called the "control string" (or "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 nothing but an array of constant characters and since all arrays decay to a pointer to their first element except in a few special cases, 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.

4. **B** Notes 6.1, 9.10; In the correct answer the Right-Left rule describes the function parameter as a "pointer to **struct** par" (in C++ the keyword **struct** is optional except when defining the structure itself) and the function argument is also type "pointer to **struct** test". Like pointers to other types, a pointer to a structure object is formed by placing the address operator to the left of the expression representing that object itself.

5. **E** 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 *'o'* in *"...old"*, the second pointer must point to the *'b'* in *"...bread"*, and the third pointer must point to the *'p'* in *"...magpie"*.

6. **B** Note 8.4; Dynamic memory allocations must always be tested for success/failure.

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A7E1_MyTime.h
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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
```

```cpp
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A7E1_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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 TESTS = 3;         // how many times to run tests

//
// 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 TESTS times.
//
int main()
{
   cout.fill('0');
   for (int testCount = 0; testCount < TESTS; ++testCount)
   {
      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;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A7E1_DetermineElapsedTime.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function DetermineElapsedTime, which calculates
// and returns a pointer to the difference between the times pointed
// to by its parameters.
//

//
// The DetermineElapsedTime function determines the amount of time elapsed
// between the times stored in the MyTime structures in <start> and <stop>
// (starting with the time in <start>) and stores 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.  A
// pointer to <elapsed> is returned.
//
// Two versions of the function are provided.  The first version uses the
// hours, minutes, and seconds directly, whereas the second version converts
// the two times to seconds and extracts the elapsed hours, minutes, and
// seconds from that.  The first version is cleaner and more efficient because
// it requires no multiplication, division, or int overflow considerations,
// and requires fewer variables.
//

#include "C1A7E1_MyTime.h"

#if 1    // Version 1

const int SEC_MIN = 60;    // seconds per minute
const int MIN_HR = 60;     // minutes per hour
const int HR_DAY = 24;     // hours per day

MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop)
{
   static MyTime elapsed;

   elapsed.hours = stop->hours - start->hours;        // hours difference
   elapsed.minutes = stop->minutes - start->minutes;  // minutes difference
   elapsed.seconds = stop->seconds - start->seconds;  // seconds difference

   if (elapsed.seconds < 0)      // borrow a minute if seconds difference is <0
   {
      elapsed.seconds += SEC_MIN;
      --elapsed.minutes;
   }

   if (elapsed.minutes < 0)      // borrow an hour if minutes difference is <0
   {
      elapsed.minutes += MIN_HR;
      --elapsed.hours;
   }

   // Add a day if stop time <= start time.
   if (elapsed.hours < 0 ||
      elapsed.hours == 0 && elapsed.minutes == 0 && elapsed.seconds == 0)
   {
      elapsed.hours += HR_DAY;
   }

   return &elapsed;              // return pointer to elapsed time structure
}

#else    // Version 2

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

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_MIN + start->seconds;
   stopSec = stop->hours * SEC_HR + stop->minutes * SEC_MIN + 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_MIN);     // minutes
   difference %= SEC_MIN;                           // seconds left
   elapsed.seconds = int(difference);               // seconds

   return &elapsed;                                 // return structure pointer
}

#endif
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A7E2_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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_QTY 5         // total lunches
#define FIXED_QTY 2         // items initialized during declaration
#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)
{
   //
   // Define type struct Food, declare an array of struct Food objects, and
   // explicitly initialize the first FIXED_QTY elements.
   //
   struct Food
   {
      char *name;
      int weight, calories;
   } lunches[LUNCH_QTY] = {{"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.
   //
   printf("Enter a space-separated food, weight, and calories...\n");
   for (int lunchIx = FIXED_QTY; lunchIx < LUNCH_QTY; ++lunchIx)
   {
      size_t length;
      char buf[BUFSIZE];                 // for getting name of food
      printf(">>> ");
      scanf(BUFFMT "s %i %i", buf, &lunches[lunchIx].weight,
         &lunches[lunchIx].calories);
      length = strlen(buf) + 1;          // characters used in buf + '\0'

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

   printf("\n"
      "         LUNCH MENU\n"
      "ITEM          WEIGHT  CALORIES\n");
   for (int lunchIx = 0; lunchIx < LUNCH_QTY; lunchIx++)
   {
      printf("%-13s%4i%10i\n", lunches[lunchIx].name,
         lunches[lunchIx].weight, lunches[lunchIx].calories);
      if (lunchIx >= FIXED_QTY)
         free(lunches[lunchIx].name);
   }
   return EXIT_SUCCESS;
}
```