Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 162461, Ray Mitchell
June 25, 2019
C1A7E0_Quiz.txt
Quiz Answers

1. B
2. D
3. B
4. D
5. E
6. A

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. **B** Notes 6.1, 9.10; In the correct answer the Right-Left rule describes the function parameter as a "pointer to **struct** test" and the function argument is also type "pointer to **struct** test". Like pointers to other types, a pointer to a structure is formed by placing the address operator to the left of the expression representing that structure itself.

2. **D** 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.

3. **B** 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.

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

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 '*b*' in "*…butcher*", the second pointer must point to the '*t*' in "*the baker*", and the third pointer must point to the '*s*' in "*…candlestick…*".

6. **A** 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 *p += 3* changed the value of *p* and the new value no longer represents the allocation address.

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, Ray Mitchell
6   // June 25, 2019
7   // C1A7E1_MyTime.h
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains the definition of structure type MyTime and a prototype
12  // for the DetermineElapsedTime function.
13  //
14
15  #ifndef C1A7E1_MYTIME_H
16  #define C1A7E1_MYTIME_H
17
18  // Define structure type to represent a time.
19  struct MyTime { int hours, minutes, seconds; };
20
21  MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop);
22
23  #endif
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, 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

static void DisplayTime(const MyTime *time);

//
// 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;
      char delim;

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

      // Determine the time difference between the two times.
      MyTime *elapsed = DetermineElapsedTime(&start, &stop);
      cout << "The time elapsed from ";
      DisplayTime(&start);
      cout << " to ";
      DisplayTime(&stop);
      cout << " is ";
      DisplayTime(elapsed);
      cout << '\n';
   }
   return EXIT_SUCCESS;
}
```

```
62
63  // Display the time in parameter "time" in HH:MM:SS format.
64  static void DisplayTime(const MyTime *time)
65  {
66     cout <<
67        setw(2) << time->hours << ":" <<
68        setw(2) << time->minutes << ":" <<
69        setw(2) << time->seconds;
70  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, Ray Mitchell
6   // June 25, 2019
7   // C1A7E1_DetermineElapsedTime.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function DetermineElapsedTime, which calculates and
12  // returns a pointer to the difference between the times pointed to by its
13  // parameters.
14  //
15
16  //
17  // The DetermineElapsedTime function determines the amount of time elapsed
18  // between the times stored in the MyTime structures in <start> and <stop>
19  // (starting with the time in <start>) and stores the result in MyTime structure
20  // <elapsed>.  If the time in <start> is greater than the time in <stop> the
21  // time in <stop> is considered to be in the next day.  A pointer to <elapsed>
22  // is returned.
23  //
24  // Two versions of the function are provided.  The first version uses the hours,
25  // minutes, and seconds directly, whereas the second version converts the two
26  // times to seconds and extracts the elapsed hours, minutes, and seconds from
27  // that.  The first version is cleaner and more efficient because it requires no
28  // multiplication, division, or int overflow considerations, and requires fewer
29  // variables.
30  //
31
32  #include "C1A7E1_MyTime.h"
33
34  #if 1                       // Version 1
35
36  const int SEC_MIN = 60;     // seconds per minute
37  const int MIN_HR = 60;      // minutes per hour
38  const int HR_DAY = 24;      // hours per day
39
40  MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop)
41  {
42     static MyTime elapsed;
43
44     elapsed.hours = stop->hours - start->hours;        // hours difference
45     elapsed.minutes = stop->minutes - start->minutes;  // minutes difference
46     elapsed.seconds = stop->seconds - start->seconds;  // seconds difference
47
48     if (elapsed.seconds < 0)      // borrow a minute if seconds difference is <0
49     {
50        elapsed.seconds += SEC_MIN;
51        --elapsed.minutes;
52     }
53
54     if (elapsed.minutes < 0)      // borrow an hour if minutes difference is <0
55     {
56        elapsed.minutes += MIN_HR;
57        --elapsed.hours;
58     }
59
60     // Add a day if stop time <= start time.
61     if (elapsed.hours < 0 ||
```

```
62          elapsed.hours == 0 && elapsed.minutes == 0 && elapsed.seconds == 0)
63       {
64          elapsed.hours += HR_DAY;
65       }
66
67       return &elapsed;              // return pointer to elapsed time structure
68    }
69
70    #else                                 // Version 2
71
72    const long SEC_MIN = 60;              // seconds per minute
73    const long SEC_HR = 60 * SEC_MIN;     // seconds per hour
74    const long SEC_DAY = 24 * SEC_HR;     // seconds per day, must be long
75
76    MyTime *DetermineElapsedTime(const MyTime *start, const MyTime *stop)
77    {
78       long startSec, stopSec, difference;
79       static MyTime elapsed;
80
81       // convert argument times into seconds
82       startSec = start->hours * SEC_HR + start->minutes * SEC_MIN + start->seconds;
83       stopSec = stop->hours * SEC_HR + stop->minutes * SEC_MIN + stop->seconds;
84
85       difference = stopSec - startSec;              // seconds elapsed
86       if (difference <= 0)                          // time is in next day
87          difference += SEC_DAY;                     // add day of seconds
88
89       // convert difference back to hours, minutes, seconds
90       elapsed.hours = int(difference / SEC_HR);     // hours
91       difference %= SEC_HR;                         // seconds left
92       elapsed.minutes = int(difference / SEC_MIN);  // minutes
93       difference %= SEC_MIN;                        // seconds left
94       elapsed.seconds = int(difference);            // seconds
95
96       return &elapsed;                              // return structure pointer
97    }
98
99    #endif
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, Ray Mitchell
6   // June 25, 2019
7   // C1A7E2_main.c
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which prompts the user for information
12  // about several foods, stores that information in memory, then displays a table
13  // containing that information.
14  //
15
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <string.h>
19
20  #define LUNCH_QTY 5         // total lunches
21  #define FIXED_QTY 2         // items initialized during declaration
22  #define BUFSIZE 256         // size of input buffer
23  #define BUFFMT "%255"       // scanf field width for buffer
24
25  //
26  // Prompt the user to input information about some food items and store them in
27  // structures in an array that already has some hard-coded food item information
28  // stored in it.  The food name strings occupy only exactly the amount of memory
29  // necessary to hold them (including the null terminator character).  After the
30  // food item information has been input and stored all food item information in
31  // the array is displayed and all dynamically- allocated memory is freed.
32  //
33  int main(void)
34  {
35     //
36     // Define type struct Food, declare an array of struct Food objects, and
37     // explicitly initialize the first FIXED_QTY elements.
38     //
39     struct Food
40     {
41        char *name;
42        int weight, calories;
43     } lunches[LUNCH_QTY] = {{"apple", 4, 100}, {"salad", 2, 80}};
44
45     //
46     // Since members of the remaining structures have not been explicitly
47     // initialized, each name member is a null pointer.  Each must, thus, be
48     // initialized to point to an area of storage where the incoming characters
49     // of the food name can be stored.
50     //
51     printf("Enter a space-separated food, weight, and calories...\n");
52     for (int lunchIx = FIXED_QTY; lunchIx < LUNCH_QTY; ++lunchIx)
53     {
54        size_t length;
55        char buf[BUFSIZE];                  // for getting name of food
56        printf(">>> ");
57        scanf(BUFFMT "s %i %i", buf, &lunches[lunchIx].weight,
58           &lunches[lunchIx].calories);
59        length = strlen(buf) + 1;           // characters used in buf + '\0'
60
61        // allocate storage for the name pointer to point to
```

```
62          if ((lunches[lunchIx].name = malloc(length)) == NULL)
63          {
64             fprintf(stderr, "malloc out of memory\n");
65             exit(EXIT_FAILURE);
66          }
67          // copy food into malloc buffer
68          memcpy(lunches[lunchIx].name, buf, length);
69       }
70
71       printf("\n"
72          "          LUNCH MENU\n"
73          "ITEM           WEIGHT  CALORIES\n");
74       for (int lunchIx = 0; lunchIx < LUNCH_QTY; lunchIx++)
75       {
76          printf("%-13s%4i%10i\n", lunches[lunchIx].name,
77             lunches[lunchIx].weight, lunches[lunchIx].calories);
78          if (lunchIx >= FIXED_QTY)
79             free(lunches[lunchIx].name);
80       }
81       return EXIT_SUCCESS;
82    }
```