Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 146359, Ray Mitchell
June 25, 2019
C1A5E0_Quiz.txt
Quiz Answers

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

C1A5E0 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. **C** Note 6.1; Determining the correct answer is simply a matter of using the "Right-Left" rule.

2. **B** Note 1.1; Neither C nor C++ have array boundary checking. As a result, if an array element is accessed that is not actually in that array, such as by using a negative index value or an index value greater than the total number of elements minus 1, a region of memory will be accessed that the code has no right to access. The result can be anything from crashing the program entirely if you are extremely lucky (which usually doesn't happen) to getting an apparently good value (that is nonetheless a garbage value) if you are extremely unlucky (which usually does happen). By definition a "garbage" value is any value (including 0) that is unpredictable or obtained via an illegal operation.

3. **B** Note 5.11; The most important thing that can be said about any code concerns issues that can cause erratic operation and/or erroneous results. In this code pointer variable *p* is declared but never initialized, and as a result will contain a garbage address. Thus, when **p = 47E3* gets executed *47E3* will be stored into the garbage address represented by *p*. Depending upon what that address actually represents, this can cause program crashes, program data corruption, or no bad effects at all.

4. **E** Notes 6.5, 6.6, 6.7, 6.9, 6.10; When used as a unary operator the address operator, **&**, produces the address of (pointer to) its operand. In the expression *fcnA(&x, &y)* the addresses of variables *x* and *y* are being passed as arguments. Although it is important to realize that addresses and pointers are one in the same thing, it is more common to say that pointers are being passed.

5. **D** Note 6.10; Either of a pointer to a non-**const** or a pointer to a **const** may be passed to a function whose corresponding parameter is a pointer to a **const**. However, only a pointer to a non-**const** may be passed to a function whose corresponding parameter is a pointer to a non-**const**. The left argument of *fcnA(&x, &y)* violates the last part of this requirement.

6. **D** Notes 5.11, 6.12; An automatic variable is any variable declared inside a block without either of the keywords **static** or **extern**. Although automatic variables cease to exist when a function returns it is a perfectly legal and common practice to return one from a function since a copy of it is made before it's destroyed. However, it is never legal to return a pointer or reference to an automatic variable since doing so returns a pointer or reference to a variable that will no longer exist after the return. The automatic variable *x* in this question is declared inside the block that forms the body of the function and it becomes invalid when the function returns. Thus, any pointers or references to it also become invalid at that time.

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A5E1_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts the user for values for
// a survey and displays the results.
//

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

#define MAX_RESPONDENTS     17   // maximum respondents
#define MIN_RESPONSE_VALUE (-27) // minimum response value
#define MAX_RESPONSE_VALUE   8   // maximum response value
#define OUT_OF_RANGE_LIMIT   1   // # of bad responses until terminate

// Number of possible response values
#define RESPONSE_VALUES (MAX_RESPONSE_VALUE - MIN_RESPONSE_VALUE + 1)

//
// Prompt user up to MAX_RESPONDENTS times to input an integral value in
// the range [MIN_RESPONSE_VALUE, MAX_RESPONSE_VALUE], keeping count of
// how many times each response value has been input.  MIN_RESPONSE_VALUE
// and MAX_RESPONSE_VALUE may be negative.  Terminate prompting and display
// the number of each possible response value that has occurred when either
// MAX_RESPONDENTS "in range" legal responses have occurred or when
// OUT_OF_RANGE_LIMIT "out-of-range" responses in a row have occurred.
//
int main(void)
{
   printf(
      "At each prompt enter an integer value from %d through %d.\n"
      "%d consecutive out-of-range entries will terminate...\n",
      MIN_RESPONSE_VALUE, MAX_RESPONSE_VALUE, OUT_OF_RANGE_LIMIT);

   int ratingCounters[RESPONSE_VALUES] = {0};
   int consecutiveRangeErrors = 0;
   // Loop to get user responses.
   for (int respondentNo = 1; respondentNo <= MAX_RESPONDENTS;)
   {
      int response;
      printf("   Respondent %d", respondentNo);          // start prompt user
      if (consecutiveRangeErrors)                        // if not user's first
         printf(" again");                               // indicate repeat
      printf(": ");                                      // end prompt user
      scanf("%d", &response);                            // get response
      // Test for type of response.
      if (response < MIN_RESPONSE_VALUE || response > MAX_RESPONSE_VALUE)
      {
         printf("      Out of range: %d\n", response);
         if (++consecutiveRangeErrors >= OUT_OF_RANGE_LIMIT)
            break;
      }
      else                                               // legal response
      {
```

```
62             ++ratingCounters[response - MIN_RESPONSE_VALUE];// inc rating count
63             ++respondentNo;                                 // next respondent
64             consecutiveRangeErrors = 0;                     // no bad responses
65          }
66       }
67       if (consecutiveRangeErrors < OUT_OF_RANGE_LIMIT)
68          printf("--Survey complete: Normal termination--\n");
69       else
70          printf("--Survey incomplete: Range error termination--\n");
71
72       // Display a table header then loop to display the number of occurrences
73       // of each rating.
74       printf("\n"
75          "Rating       Responses\n"
76          "------       ---------\n");
77       for (int responseIx = RESPONSE_VALUES - 1; responseIx >= 0; --responseIx)
78          printf("%4d%14d\n",
79             responseIx + MIN_RESPONSE_VALUE, ratingCounters[responseIx]);
80
81       return EXIT_SUCCESS;
82    }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E2_main.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which prompts the user for two values,
12  // calls functions that determine their maximum and minimum using references,
13  // then displays the results.
14  //
15
16  #include <iostream>
17  using std::cin;
18  using std::cout;
19
20  // Prototypes for custom functions called by main.
21  double &ComputeMaximum(const double &val1, const double &val2);
22  double &ComputeMinimum(const double &val1, const double &val2);
23
24  //
25  // Test functions ComputeMinimum and ComputeMaximum by calling them with
26  // values obtained from user input and display the results.  These functions
27  // must return references to the minimum and maximum of the two values whose
28  // references are passed to them as arguments.
29  //
30  int main()
31  {
32     double value1, value2;
33     // Get two user input values and determine which is lesser/greater.
34     cout << "Enter two space-separated decimal values: ";
35     cin >> value1 >> value2;
36
37     cout << "ComputeMinimum(" << value1 << ", " << value2 << ") returned " <<
38        ComputeMinimum(value1, value2) << "\n";
39     cout << "ComputeMaximum(" << value1 << ", " << value2 << ") returned " <<
40        ComputeMaximum(value1, value2) << "\n";
41
42     return 0;
43  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E2_ComputeMaximum.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function ComputeMaximum, which returns a reference
12  // to the maximum of the two values referenced by its parameters.
13  //
14
15  //
16  // Determine the maximum of the values referenced by its two parameters and
17  // return a reference to it.  No special test is needed for equal values.
18  //
19  double &ComputeMaximum(const double &val1, const double &val2)
20  {
21     // Compare <val1> and <val2> and return the maximum.
22     return (double &)(val1 > val2 ? val1 : val2);
23  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E2_ComputeMinimum.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function ComputeMinimum, which returns a reference
12  // to the minimum of the two values referenced by its parameters.
13  //
14
15  //
16  // Determine the minimum of the values referenced by its two parameters and
17  // return a reference to it.  No special test is needed for equal values.
18  //
19  double &ComputeMinimum(const double &val1, const double &val2)
20  {
21     // Compare <val1> and <val2> and return the minimum.
22     return (double &)(val1 < val2 ? val1 : val2);
23  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E3_main.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which prompts the user for two values,
12  // calls functions that determine their maximum and minimum using pointers,
13  // then displays the results.
14  //
15
16  #include <iostream>
17  using std::cin;
18  using std::cout;
19
20  // Prototypes for custom functions called by main.
21  double *ComputeMaximum(const double *val1, const double *val2);
22  double *ComputeMinimum(const double *val1, const double *val2);
23
24  //
25  // Test functions ComputeMinimum and ComputeMaximum by calling them with
26  // values obtained from user input and display the results.  These functions
27  // must return pointers to the minimum and maximum of the two values whose
28  // pointers are passed to them as arguments.
29  //
30  int main()
31  {
32     double value1, value2;
33     // Get two user input values and determine which is lesser/greater.
34     cout << "Enter two space-separated decimal values: ";
35     cin >> value1 >> value2;
36
37     cout << "ComputeMinimum(&" << value1 << ", &" << value2 << ") returned &" <<
38        *ComputeMinimum(&value1, &value2) << "\n";
39     cout << "ComputeMaximum(&" << value1 << ", &" << value2 << ") returned &" <<
40        *ComputeMaximum(&value1, &value2) << "\n";
41
42     return 0;
43  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E3_ComputeMaximum.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function ComputeMaximum, which returns a pointer
12  // to the maximum of the two values pointed to by its parameters.
13  //
14
15  //
16  // Determine the maximum of the values pointed to by its two parameters and
17  // return a pointer to it. No special test is done for the values being equal.
18  //
19  double *ComputeMaximum(const double *val1, const double *val2)
20  {
21     // Compare <*val1> and <*val2> and return the maximum.
22     return (double *)(*val1 > *val2 ? val1 : val2);
23  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A5E3_ComputeMinimum.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function ComputeMinimum, which returns a pointer
12  // to the minimum of the two values pointed to by its parameters.
13  //
14
15  //
16  // Determine the minimum of the values pointed to by its two parameters and
17  // return a pointer to it. No special test is done for the values being equal.
18  //
19  double *ComputeMinimum(const double *val1, const double *val2)
20  {
21     // Compare <*val1> and <*val2> and return the minimum.
22     return (double *)(*val1 < *val2 ? val1 : val2);
23  }
```