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

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E1_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts shoppers to input survey
// ratings and displays a table of the results.
//

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

#define SHOPPERS 17                             // number of shoppers
#define BEST 8                                  // best rating value
#define WORST (-27)                             // worst rating value
#define CHOICES (BEST - WORST + 1)              // # of unique rating values
#define MAX_ERRORS 1          // # of out-of-range ratings required to terminate

//
// This function prompts up to SHOPPERS shoppers to enter an integer rating
// value in the range [WORST, BEST] and keeps count of how many times each
// rating value has been entered.  WORST must always be <= BEST and BEST may be
// negative, zero, or positive. When either all shoppers have entered an
// in-range rating or any one shopper enters MAX_ERRORS out-of-range values, the
// function displays a table of survey results and returns.
//
int main(void)
{
   printf(
      "At each prompt enter an integer value from %d through %d, OR\n"
      "enter %d out-of-range values to terminate...\n",
      WORST, BEST, MAX_ERRORS);

   int ratings[CHOICES] = {0};                  // holds rating occurrence counts
   int errors = 0;
   // Loop to get shopper ratings.
   for (int shopperNo = 1; shopperNo <= SHOPPERS;)
   {
      int rating;
      printf("   Shopper %d", shopperNo);       // start shopper prompt
      if (errors)                               // if not shoppers's 1st error
         printf(" again");                      // indicate repeat
      printf(": ");                             // end shopper prompt

      scanf("%d", &rating);                     // get rating
      // Test for valid rating.
      if (rating < WORST || rating > BEST)
      {
         printf("      Out of range: %d\n", rating);
         if (++errors >= MAX_ERRORS)
            break;
      }
      else                                      // legal rating
      {
         ++ratings[rating - WORST];             // increment rating count
         ++shopperNo;                           // next shopper
```

```
            errors = 0;                             // reset bad ratings count
         }
      }
      if (errors < MAX_ERRORS)
         printf("--Survey complete: Normal termination--\n");
      else
         printf("--Survey incomplete: Range error termination--\n");

      // Display a table header then loop to display the number of occurrences of
      // each possible allowed rating.
      printf("\n"
         "Rating    Quantity\n"
         "------    --------\n");
      for (int ratingIx = CHOICES - 1; ratingIx >= 0; --ratingIx)
         printf("%6d%12d\n", ratingIx + WORST, ratings[ratingIx]);

      return EXIT_SUCCESS;
   }
```

```cpp
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E2_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts the user for two values,
// calls functions that determine their maximum and minimum using references,
// then displays the results.
//

#include <iostream>
using std::cin;
using std::cout;

// Prototypes for custom functions called by main.
double &ComputeMaximum(const double &n1, const double &n2);
double &ComputeMinimum(const double &n1, const double &n2);

//
// Test functions ComputeMinimum and ComputeMaximum by calling them with values
// obtained from user input and display the results.  These functions must
// return references to the minimum and maximum of the two values whose
// references are passed to them as arguments.
//
int main()
{
   // Prompt for and get two user input values.
   double in1, in2;
   cout << "Enter two space-separated decimal values: ";
   cin >> in1 >> in2;

   // Determine and display which value is lesser/greater.
   cout << "ComputeMinimum(" << in1 << ", " << in2 << ") returned " <<
      ComputeMinimum(in1, in2) << "\n";
   cout << "ComputeMaximum(" << in1 << ", " << in2 << ") returned " <<
      ComputeMaximum(in1, in2) << "\n";

   return 0;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E2_ComputeMaximum.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function ComputeMaximum, which returns a reference to the
// highest of the two values referenced by its parameters.
//

double &ComputeMaximum(const double &n1, const double &n2)
{
   return (double &)(n1 > n2 ? n1 : n2);
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E2_ComputeMinimum.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function ComputeMinimum, which returns a reference to the
// lowest of the two values referenced by its parameters.
//

double &ComputeMinimum(const double &n1, const double &n2)
{
   return (double &)(n1 < n2 ? n1 : n2);
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E3_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts the user for two values,
// calls functions that determine their maximum and minimum using pointers,
// then displays the results.
//

#include <iostream>
using std::cin;
using std::cout;

// Prototypes for custom functions called by main.
double *ComputeMaximum(const double *p1, const double *p2);
double *ComputeMinimum(const double *p1, const double *p2);

//
// Test functions ComputeMinimum and ComputeMaximum by calling them with values
// obtained from user input and display the results.  These functions must
// return pointers to the minimum and maximum of the two values whose pointers
// are passed to them as arguments.
//
int main()
{
   // Prompt for and get two user input values.
   double in1, in2;
   cout << "Enter two space-separated decimal values: ";
   cin >> in1 >> in2;

   // Determine and display which value is lesser/greater.
   cout << "ComputeMinimum(&" << in1 << ", &" << in2 << ") returned &" <<
      *ComputeMinimum(&in1, &in2) << "\n";
   cout << "ComputeMaximum(&" << in1 << ", &" << in2 << ") returned &" <<
      *ComputeMaximum(&in1, &in2) << "\n";

   return 0;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E3_ComputeMaximum.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function ComputeMaximum, which returns a pointer to the
// highest of the two values pointed to by its parameters.
//

double *ComputeMaximum(const double *p1, const double *p2)
{
   return (double *)(*p1 > *p2 ? p1 : p2);
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A5E3_ComputeMinimum.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function ComputeMinimum, which returns a pointer to the
// lowest of the two values pointed to by its parameters.
//

double *ComputeMinimum(const double *p1, const double *p2)
{
   return (double *)(*p1 < *p2 ? p1 : p2);
}
```