```
Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 162461, Ray Mitchell
June 25, 2019
C1A2E0_Quiz.txt
Quiz Answers

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

C1A2E0 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 2.3; Although the number of bits in a byte is certainly 8 on almost all systems, this is not required by the C and C++ language standards. In fact, the old UNIVAC 11xx series computers used 9-bit bytes while other systems commonly used 6-bit bytes.

2. **D** Notes 2.1 & 2.4; Type **int** is typically the most time-efficient integer type for numeric calculations while type **double** is the most common type in the standard floating point library.

3. **B** Notes 2.1, 2.2, & 2.4; The data type of an integer literal is determined by its value, base, and suffix (if any). The data type of a floating literal is determined entirely by its suffix. Non-suffixed floating literals are type **double**.

4. **D** Note 2.10; In any arithmetic operation involving more than one operand, subinteger operands are first promoted to type **int** or **unsigned int**.

5. **A** Note 2.13; There are two main good practice guidelines regarding the syntax of the replacement lists of almost all macros:
   1) If the replacement list consists of more than one token parenthesize the replacement list.
   2) If the macro has arguments parenthesize each argument instance in the replacement list.

   Guideline 1 applies in the case of *#define SUM 5+3* but was not followed. Thus, although the value of the expression *6*SUM* should be 48 it is instead 33 because the value of SUM is not 8. That is,
   *6*SUM = 6*5+3 = 33*
   If the macro had been written properly as *#define SUM (5+3)* then,
   *6*SUM = 6*(5+3) = 48*

6. **C** Note 2.12; The data type produced by the **sizeof** operator is implementation dependent but must be one of the unsigned integer types (**unsigned char**, **unsigned short**, **unsigned int**, **unsigned long**, or **unsigned long long**). The *printf %u* conversion specification is only compatible with types **unsigned char**, **unsigned short**, and **unsigned int**.

```cpp
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A2E1_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which converts a user input character to
// lowercase.
//

#include <iostream>
#include <cstdlib>

const int CASE_DIFF = 'a' - 'A'; // assumed constant lowercase/uppercase diff.

//
// Convert the character input by the user to lowercase by adding the numeric
// difference between the lowercase and uppercase character sets to the value of
// the user input character. If a non-uppercase character is input the result
// will be the character having the new value or implementation dependent if
// there is no such character. This algorithm assumes that the distance between
// corresponding members of the lowercase and uppercase character sets is the
// same for all members. That is, 'a'-'A' == 'b'-'B' == 'c'-'C', etc. The only
// appropriate and truly portable solution would be to use the tolower function
// to do the conversion, but that technique was not allowed in this exercise.
//
int main()
{
   // Get user input character, convert, then output result.
   std::cout << "Enter an uppercase character: ";
   char ch = (char)std::cin.get();
   std::cout << "The lowercase equivalent of '" << ch
      << "' is '" << (char)(ch + CASE_DIFF) << "'\n";

   return EXIT_SUCCESS;
}
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A2E2_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts the user for a value and
// displays that number of lines to form a triangle of characters.
//

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

#define LEADER_CHAR '#'
#define DIAGONAL_CHAR '$'

//
// Display the character specified by DIAGONAL_CHAR diagonally on the number of
// lines specified by user input.  On the last line DIAGONAL_CHAR will be in the
// first column, on the next to last line it will be in the second column, etc.
// On each line DIAGONAL_CHAR will be preceded by the number of copies of the
// character specified by LEADER_CHAR as necessary to reach the column where
// DIAGONAL_CHAR is to be displayed.  For example, if the user entered 4 and
// LEADER_CHAR were ^ and DIAGONAL_CHAR were @, the output would be:
// ^^^@
// ^^@
// ^@
// @
//
int main(void)
{
   int lines;

   printf("Enter a line count: ");
   scanf("%d", &lines);                               // get user line count
   for (int lineNo = 0; lineNo < lines; ++lineNo)  // line loop
   {
      // column loop
      for (int leadChars = lineNo + 1; leadChars < lines; ++leadChars)
         putchar(LEADER_CHAR);                  // print leader value
      printf("%c\n", DIAGONAL_CHAR);          // print diagonal char & '\n'
   }
   return EXIT_SUCCESS;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A2E3_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which prompts the user for a value and
// displays that number of lines to form a triangle of characters.
//

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

const char LEADER_CHAR = '#';
const char DIAGONAL_CHAR = '$';

//
// Display the character specified by DIAGONAL_CHAR diagonally on the number of
// lines specified by user input.  On the last line DIAGONAL_CHAR will be in the
// first column, on the next to last line it will be in the second column, etc.
// On each line DIAGONAL_CHAR will be preceded by the number of copies of the
// character specified by LEADER_CHAR as necessary to reach the column where
// DIAGONAL_CHAR is to be displayed.  For example, if the user entered 4 and
// LEADER_CHAR were ^ and DIAGONAL_CHAR were @, the output would be:
// ^^^@
// ^^@
// ^@
// @
//
int main()
{
   int lines;

   cout << "Enter a line count: ";
   cin >> lines;                                  // get user line count
   for (int lineNo = 0; lineNo < lines; ++lineNo)  // line loop
   {
      // column loop
      for (int leadChars = lineNo + 1; leadChars < lines; ++leadChars)
         cout << LEADER_CHAR;                  // print leader value
      cout << DIAGONAL_CHAR << '\n';           // print diagonal char & '\n'
   }
   return EXIT_SUCCESS;
}
```