Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 146359, Ray Mitchell
June 25, 2019
C1A2E0_Quiz.txt
Quiz Answers

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

## 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** Notes 2.10, 2.11; In all cases other than answer E the multiplication of 2 and 16384, which are both of type **int** on any and every machine, will be done using type **int** math. In implementations where the range of type **int** is at the ANSI minimum of ±32767 (this requires at least 16 bits), the potential answer of 32768 will not fit, overflow will occur, and a garbage answer will result. However, if 16384**L** (which has type **long**) is used instead, the 2 will automatically be converted to type **long** to match the type of the 16384**L** and the math will be done using type **long**. Since the ANSI minimum range for type **long** is ±2147483647 (this requires at least 32 bits), the correct answer of 3276800 will easily fit.

2. **C** Notes 2.1, 2.2A, 2.2B, 2.11; The data type of an integer literal is determined by its value, base, and suffix (if any) according to the table in note 2.2A. If the value of a non-suffixed decimal integer literal is not too great to be represented as type **int** it will be type **int**. Otherwise it will have type **long** or **long long**. In implementations where the range of type **int** is at the ANSI minimum of ±32767 (this requires at least 16 bits), 32767 will be type **int** while 32768 will be type **long**. In most modern implementations the range of type **int** is much greater and 32768 will also be type **int**.

3. **E** Notes 2.2A, 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. **B** Note 2.8; Integer division discards the remainder, regardless of its value, and keeps the whole part. Floating division keeps both parts.

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

6. **D** The value of a **sizeof** expression is determined only by the data type of its operand, which is type **int** in both cases in this question.

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A2E1_main.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which converts a user input character
12  // to lowercase.
13  //
14
15  #include <iostream>
16  #include <cstdlib>
17
18  const int CASE_DIFF = 'a' - 'A'; // assumed constant lower-upper case diff.
19
20  //
21  // Convert the character input by the user to lowercase by adding the numeric
22  // difference between the lowercase and uppercase character sets to the value
23  // of the user input character. If a non-uppercase character is input the
24  // result will be the character having the new value or implementation
25  // dependent if there is no such character. This algorithm assumes that
26  // the distance between corresponding members of the lowercase and uppercase
27  // character sets is the same for all members. That is, 'a'-'A' == 'b'-'B'
28  // == 'c'-'C', etc. The only appropriate and truly portable solution would
29  // be to use the tolower function to do the conversion, but that technique
30  // was not allowed in this exercise.
31  //
32  int main()
33  {
34     // Get user input character, convert, then output result.
35     std::cout << "Enter an uppercase character: ";
36     char ch = (char)std::cin.get();
37     std::cout << "The lowercase equivalent of '" << ch
38        << "' is '" << (char)(ch + CASE_DIFF) << "'\n";
39
40     return EXIT_SUCCESS;
41  }
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, 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 DIAGONAL_CHAR '^'
#define BASE 10

//
// Display the character specified by DIAGONAL_CHAR diagonally on the number
// of lines specified by user input.  On the first line DIAGONAL_CHAR will be
// in the first column, on the next line it will be in the second column, etc.
// On each line DIAGONAL_CHAR will be preceded by the quantity of sequential
// 1-digit decimal values necessary to reach the column where
// DIAGONAL_CHAR is to be displayed.  The 1-digit values will start with
// the least significant digit of the user input value and increment by 1
// until 9 is reached, at which time they will start over at 0.  For example,
// for a DIAGONAL_CHAR of '@' and a user input of 5 the output would be:
// @
// 5@
// 67@
// 890@
// 1234@
//
int main(void)
{
   int lines;
   printf("Enter a line count: ");
   scanf("%d", &lines);                        // get user line count
   int leaderValue = lines;
   for (int lineNo = 0; lineNo < lines; ++lineNo)   // line loop
   {
      // column loop
      for (int leadChars = 0; leadChars < lineNo; ++leadChars)
         printf("%d", leaderValue++ % BASE); // print leader LSD
      printf("%c\n", DIAGONAL_CHAR);         // print diagonal char and '\n'
   }

   return EXIT_SUCCESS;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, 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 DIAGONAL_CHAR = '$';
const int BASE = 10;

//
// Display the character specified by DIAGONAL_CHAR diagonally on the number
// of lines specified by user input.  On the first line DIAGONAL_CHAR will be
// in the first column, on the next line it will be in the second column, etc.
// On each line DIAGONAL_CHAR will be preceded by the quantity of sequential
// 1-digit decimal values necessary to reach the column where
// DIAGONAL_CHAR is to be displayed.  The 1-digit values will start with
// the least significant digit of the user input value and increment by 1
// until 9 is reached, at which time they will start over at 0.  For example,
// for a DIAGONAL_CHAR of '@' and a user input of 5 the output would be:
// @
// 5@
// 67@
// 890@
// 1234@
//
int main()
{
   int lines;

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