Exercise 0 *(6 points – 1 point per question – No program required)*

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

References and 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. 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), 252767 will be type **long**. In most modern implementations the range of type **int** is much greater and 252767 will be type **int**.

2. 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**.

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

4. Note 2.8; When integer division is performed with one or both operands negative, two implementation dependent quotients are possible. When remainder division is performed with one or both operands negative, two implementation dependent remainders are possible since the formula used to compute the remainder, *a % b == a - (a / b) * b*, includes an implementation dependent integer division.

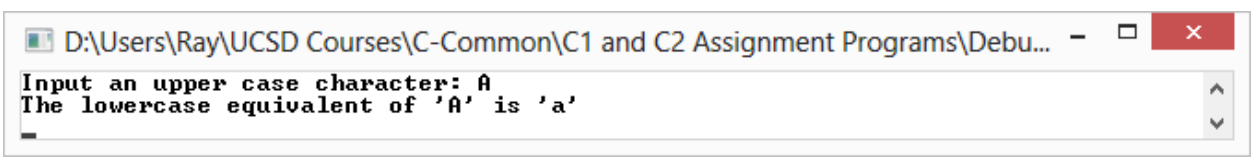
5. For *-7 % 3* the reference and explanation are the same as for question 4. See Note 2.12 for *sizeof(-5 % 3)*. The **sizeof** unary operator produces a value equal to the number of bytes in the data type of its operand after any type conversions on that operand have taken place. The value of a **sizeof** expression is determined only by the data type of its operand.

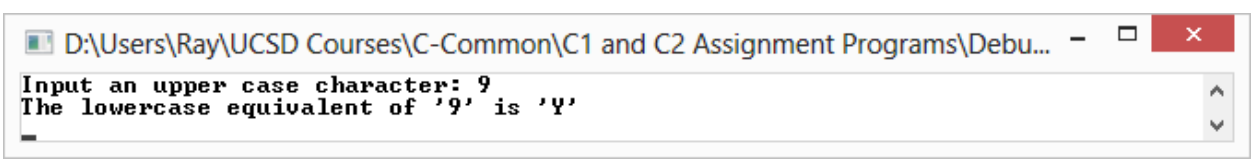
6. Notes 2.10, 2.11; In all cases other than answer B the multiplication of 300 and 400, which are both of type **int** on any and every machine, will be done using type **int** math. In implementations where the range of type **int** is at the ANSI minimum of ±32767 (this requires at least 16 bits), the potential answer of 120000 will not fit, overflow will occur, and a garbage answer will result. However, if 400**L** (which has type **long**) is used instead, the 300 will automatically be converted to type **long** to match the type of the 400**L** and the math will be done using type **long** math. Since the ANSI minimum range for type **long** is ±2147483647 (this requires at least 32 bits), the correct answer of 120000 will easily fit.

Exercise 1 *(5 points – C++ Program)*

```
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function main, which and attempts to display the
// lowercase equivalent of a user-prompted character.
//

#include <iostream>
#include <cstdlib>

const int CASE_DIFF = 'a' - 'A';    // assumed constant lower-upper case 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 << "Input an upper case character: ";
   char ch = (char)std::cin.get();
   std::cout << "The lowercase equivalent of '" << ch
      << "' is '" << (char)(ch  + CASE_DIFF) << "'\n";

   return EXIT_SUCCESS;
}
```

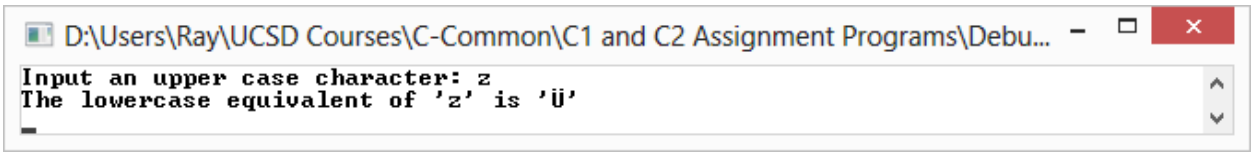
C1A2E1 Screen Shots

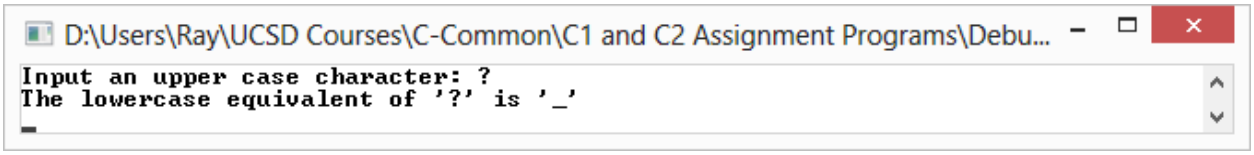




C1A2E1 Screen Shots continue on the next page…

C1A2E1 Screen Shots, continued

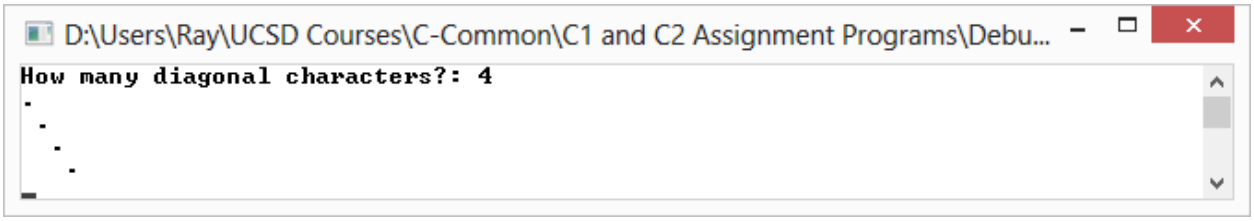




Exercise 2 *(6 points – C Program)*

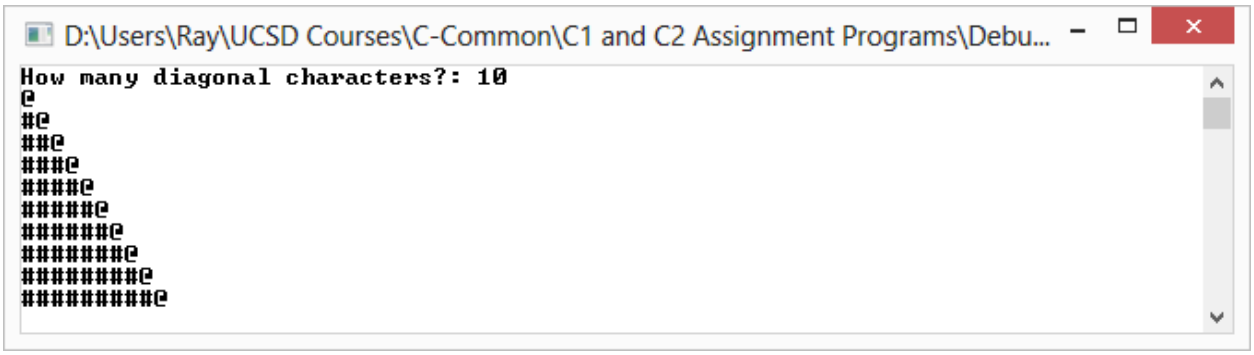
```c
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function main, which prompts the user for a value and
 * displays a diagonal line containing that number of characters.
 */

#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 first line DIAGONAL_CHAR will be in
 * the first column, on the second 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, for a user input of 4 the
 * output would be:
 * @
 * #@
 * ##@
 * ###@
 */
int main(void)
{
   int diagChars, lineNo;

   printf("How many diagonal characters?: ");
   scanf("%d", &diagChars);                  /* get user character count */
   for (lineNo = 0; lineNo < diagChars; ++lineNo)           /* line loop */
   {
      int leadChars;
      for (leadChars = 0; leadChars < lineNo; ++leadChars)  /* column loop */
         putchar(LEADER_CHAR);               /* print leader char */
      printf("%c\n", DIAGONAL_CHAR);         /* print diagonal char & '\n' */
   }
   return EXIT_SUCCESS;
}
```

C1A2E2 Screen Shots



C1A2E2 Screen Shots continue on the next page…

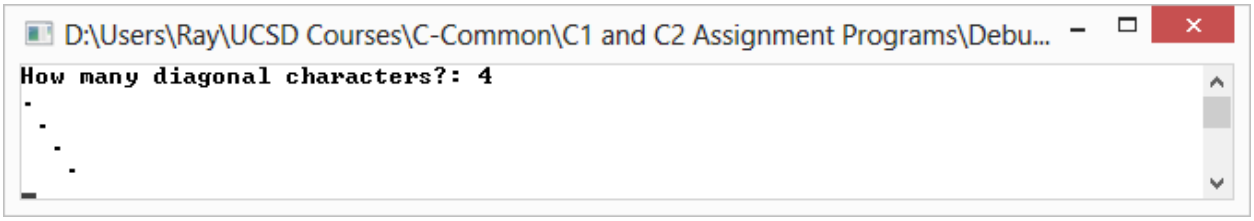
C1A2E2 Screen Shots, continued



Exercise 3 *(3 points – C++ Program)*

```
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function main, which prompts the user for a value then
// displays a diagonal line containing that number 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 first line DIAGONAL_CHAR will be in
// the first column, on the second 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, for a user input of 4 the
// output would be:
// @
// #@
// ##@
// ###@
//
int main()
{
   int diagChars;

   cout << "How many diagonal characters?: ";
   cin >> diagChars;                            // get user character count
   for (int lineNo = 0; lineNo < diagChars; ++lineNo)           // line loop
   {
      for (int leadChars = 0; leadChars < lineNo; ++leadChars)  // column loop
         cout << LEADER_CHAR;                   // print leader char
      cout << DIAGONAL_CHAR << '\n';            // print diagonal char & '\n'
   }
   return EXIT_SUCCESS;
}
```

C1A2E3 Screen Shots



C1A2E3 Screen Shots continue on the next page…

C1A2E3 Screen Shots, continued