Exercise 1 *(3 points – C Program)*

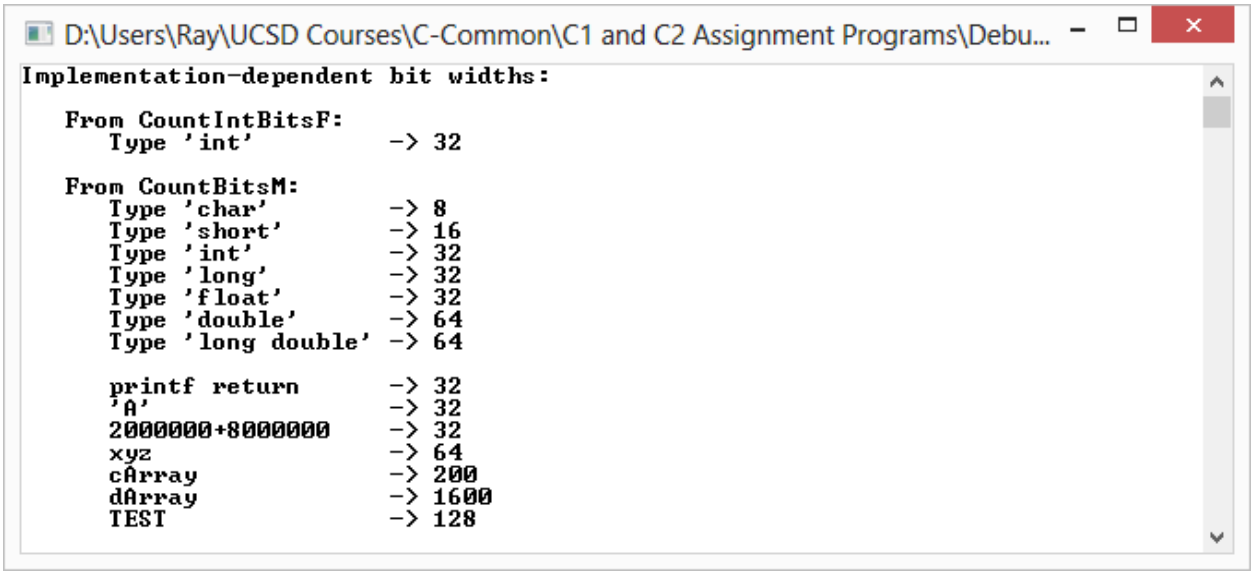
********** FILE C2A2E1_CountBitsM.h **********

```c
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains macro CountBitsM, which returns the number of bits of
 * storage in the data type of its parameter.
 */

#ifndef C2A2E1_COUNTBITSM_H
#define C2A2E1_COUNTBITSM_H

#include <limits.h>

/*
 * Macro CountBitsM produces a count of the number of bits of storage needed
 * to represent the data type of the object or data type represented by
 * parameter <objectOrType>.
 *
 * IMPORTANT:
 * Note that there is a potential problem with the following macro if it is
 * used to determine the number of bits actually used to represent a value
 * having a particular data type.  Careful consideration must be given when
 * using it for that purpose since it can give incorrect results for some
 * data types in certain implementations.  To understand why realize that
 * the sizeof operator produces a count of the number of bytes of storage
 * required to store the data type of its operand.  However, for some types
 * not all of that storage may be used to represent the object's value.  In
 * those cases one or more additional unused bits or bytes of "padding" are
 * included simply to permit proper alignment of the object in memory.
 * While this does not usually occur with most scalar types there can be
 * exceptions, with the most notable being type long double in some
 * implementations.  In some cases this data type requires 8 bytes of storage
 * and all of them are used to represent its value (no padding).  In other
 * cases, however, 16 bytes of storage are used, and only 10 or 12 of them
 * are actually used to represent its value.
 * ...Caveat Emptor...
 */
#define CountBitsM(objectOrType) ((int)sizeof(objectOrType) * CHAR_BIT)

#endif
```

---------- EXERCISE CONTINUES ON NEXT PAGE ----------

```

    ************************************ FILE C2A2E1_CountIntBitsF.c ************************************

/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function CountIntBitsF, which returns the number of bits
 * used to represent the value of type int.
 */

/*
 * Determine the number of bits used to represent any and every value having
 * data type int.  This is not necessarily the same as the number of bits of
 * memory used for type int.
 */
int CountIntBitsF(void)
{
   int bitsInInt;
   unsigned pattern;

   /*
    * Store a 1 into an unsigned int variable and repeatedly shift left until
    * the value of the variable becomes 0.  The number of shifts necessary
    * indicates the number of usable bits in the data type of that variable.
    */
   for (bitsInInt = 0, pattern = 1u; pattern; pattern <<= 1, ++bitsInInt)
      ;
   return bitsInInt;
}
```

C2A2E1 Screen Shot (Values are implementation-dependent.)



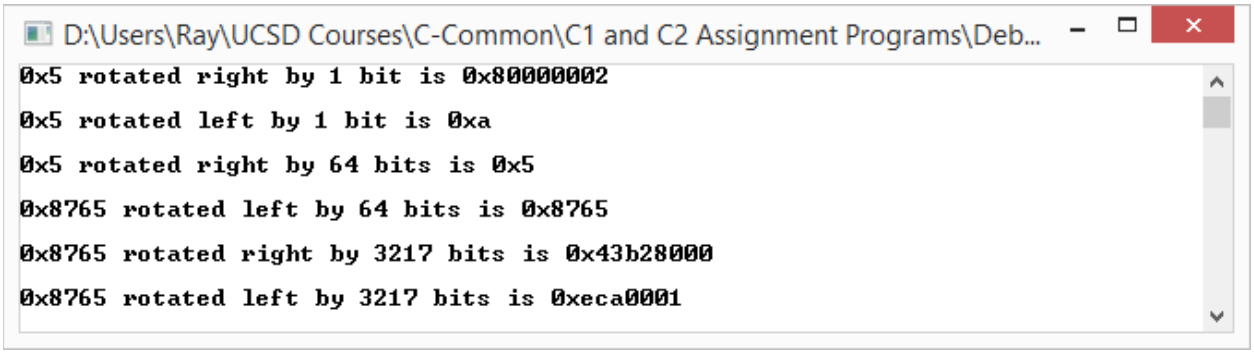
Exercise 2 *(5 points – C++ Program)*

```
   ************************************ FILE C2A2E2_CountIntBitsF.cpp ************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function CountIntBitsF, which returns the number of bits
// used to represent the value of type int.
//

//
// Determine the number of bits used to represent any and every value having
// data type int.  This is not necessarily the same as the number of bits of
// memory used for type int.
//
int CountIntBitsF()
{
   int bitsInInt;
   unsigned pattern;

   /*
    * Store a 1 into an unsigned int variable and repeatedly shift left until
    * the value of the variable becomes 0.  The number of shifts necessary
    * indicates the number of usable bits in the data type of that variable.
    */
   for (bitsInInt = 0, pattern = 1u; pattern; pattern <<= 1, ++bitsInInt)
      ;
   return bitsInInt;
}


```

---------- EXERCISE CONTINUES ON NEXT PAGE ----------

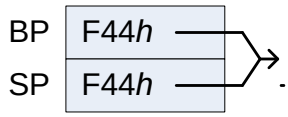
```
   ************************************** FILE C2A2E2_Rotate.cpp **************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function Rotate, which returns its first parameter
// rotated by the number of bits specified by its second parameter.
//

int CountIntBitsF();

//
// Return the value resulting from rotating the pattern in <object> by the
// number of bit positions specified by <count>.  If <count> is positive
// rotation will be to the right; if <count> is negative rotation will be
// to the left.  The Rotate function exploits the fact that a left rotation
// by <count> is equivalent to a right rotation by the total bit-width
// of the object minus <count>.  Note that the result of shifting an object
// by a negative amount or by an amount greater than the number of bits
// in the object is undefined.  The modulus operator is used to prevent
// the second case from happening.
//
unsigned Rotate(unsigned object, int count)
{
   int bits = CountIntBitsF();

   // Get (abs(count) % bits) in case (count >= bits) in object.
   if (count < 0)
      count = bits - (-count % bits);
   else
      count %= bits;

   return((object >> count) | (object << (bits - count)));
}
```

C2A2E2 Screen Shot for 32-bit **unsigned int**
(Your results will depend upon the width of your type **unsigned int**.)



Exercise 3 *(6 points – Drawing only – No program required)*

Memory Addresses: Relative	Memory Addresses: Absolute	Stack Values	Description	
BP+??	??	??	??	***startup* Stack Frame**
BP+??	FA9*h*	??	??	
BP+A*h*	FA6*h*	??	Return Object (int)	***main* Stack Frame**
BP+5*h*	FA1*h*	??	Function Return Address	
BP	F9C*h*	??	Previous Frame Address	
BP-4*h*	F98*h*	??	val	
BP+A*h*	F94*h*	??	Return Object (long)	***Ready* Stack Frame**
BP+5*h*	F8F*h*	AB4*h*	Function Return Address	
BP	F8A*h*	F9C*h*	Previous Frame Address	
BP-4*h*	F86*h*	??	res	
BP+12*h*	F82*h*	??	Return Object (long)	***gcd* Stack Frame 1**
BP+E*h*	F7E*h*	96	y	
BP+A*h*	F7A*h*	128	x	
BP+5*h*	F75*h*	108*h*	Function Return Address	
BP	F70*h*	F8A*h*	Previous Frame Address	
BP+12*h*	F6C*h*	??	Return Object (long)	***gcd* Stack Frame 2**
BP+E*h*	F68*h*	32	y	
BP+A*h*	F64*h*	96	x	
BP+5*h*	F5F*h*	7C0*h*	Function Return Address	
BP	F5A*h*	F70*h*	Previous Frame Address	
BP+12*h*	F56*h*	??	Return Object (long)	***gcd* Stack Frame 3**
BP+E*h*	F52*h*	0	y	
BP+A*h*	F4E*h*	32	x	
BP+5*h*	F49*h*	7C0*h*	Function Return Address	
BP	F44*h*	F5A*h*	Previous Frame Address	



No C2A2E3 Screen Shot – Not a program

Exercise 4 *(6 points – C++ Program)*

```
   ************************************** FILE C2A2E4_OpenFile.cpp **************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function OpenFile, which opens for input the file
// specified by its first parameter using the object specified by its
// second parameter.
//

#include <fstream>
#include <iostream>
#include <cstdlib>

//
// Open the file named in <fileName> using the object referenced by
// <inFile>.  If it fails display an error message and terminate the
// program with an error code.
//
void OpenFile(const char *fileName, std::ifstream &inFile)
{
   // Open file for read only.
   inFile.open(fileName);
   // If open fails print an error message and terminate with an error code.
   if (!inFile.is_open())
   {
      std::cerr << "File \"" << fileName << "\" didn't open.\n";
      std::exit(EXIT_FAILURE);
   }
}

    ************************************** FILE C2A2E4_Reverse.cpp **************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains functions:
//   IsSep: Determines if its parameter represents a separator;
//   Reverse: Recursively reverses and displays characters read from a file;
//

#include <cctype>
#include <fstream>
#include <iostream>

const int CAPITALIZATION_LEVEL = 1; // recursive level for capitalization

//
// Test if the value in <ch> is one of the separators required in this
// exercise.  Return true if so and false if not.
//
inline bool IsSep(int ch)
{
   //
   // Whitespace is most appropriately checked by isspace, not by
   // checking individual specific characters.
   //
```

```
   return(std::isspace(ch) || ch == '.' || ch == '?' || ch == '!' ||
      ch == ',' || ch == ':' || ch == ';' || ch == EOF);
}

//
// As each recursive level of the function is entered one input character is
// read and stored in <thisChar>.  This continues until a separator is
// encountered, which is then stored in <thisSeparator>.  The function then
// begins returning <thisSeparator> back through all levels of recursion.
// After each return the character in <thisChar> is displayed and, if at
// the level specified by CAPITALIZATION_LEVEL, is also capitalized.
// The function then returns the separator to the caller.
//
int Reverse(std::ifstream &inFile, const int level)
{
   int thisChar = inFile.get();                 // get next character...
   if (IsSep(thisChar))                         // ...if character is separator
      return thisChar;                          // ...then return it

   int thisSeparator = Reverse(inFile, level + 1); // get next character
   // Print character, capitalizing if at level CAPITALIZATION_LEVEL.
   if (level == CAPITALIZATION_LEVEL)
      std::cout.put((char)std::toupper(thisChar));
   else
      std::cout.put((char)thisChar);

   return thisSeparator;                         // return separator
}
```

C2A2E4 Screen Shot