

Exercise 1 (3 points – C Program)

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

```
/*
 * ...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 -----

```

1
2 ***** FILE C2A2E1_CountIntBitsF.c *****
3
4 /*
5  * ...the usual title block Student/Course/Assignment/Compiler information goes here...
6  *
7  * This file contains function CountIntBitsF, which returns the number of bits
8  * used to represent the value of type int.
9  */
10
11 /*
12  * Determine the number of bits used to represent any and every value having
13  * data type int. This is not necessarily the same as the number of bits of
14  * memory used for type int.
15  */
16 int CountIntBitsF(void)
17 {
18     int bitsInInt;
19     unsigned pattern;
20
21     /*
22      * Store a 1 into an unsigned int variable and repeatedly shift left until
23      * the value of the variable becomes 0. The number of shifts necessary
24      * indicates the number of usable bits in the data type of that variable.
25      */
26     for (bitsInInt = 0, pattern = 1u; pattern; pattern <<= 1, ++bitsInInt)
27         ;
28     return bitsInInt;
29 }

```

C2A2E1 Screen Shot (Values are implementation-dependent.)

The screenshot shows a debugger window titled "D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Debu...". The main content area displays "Implementation-dependent bit widths:" with the following data:

Source	Type	Value
From CountIntBitsF:	Type 'int'	-> 32
From CountBitsM:	Type 'char'	-> 8
	Type 'short'	-> 16
	Type 'int'	-> 32
	Type 'long'	-> 32
	Type 'float'	-> 32
	Type 'double'	-> 64
	Type 'long double'	-> 64
printf return		-> 32
'a'		-> 32
20000000+80000000		-> 32
xyz		-> 64
cArray		-> 200
dArray		-> 1600
TEST		-> 128

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;
}

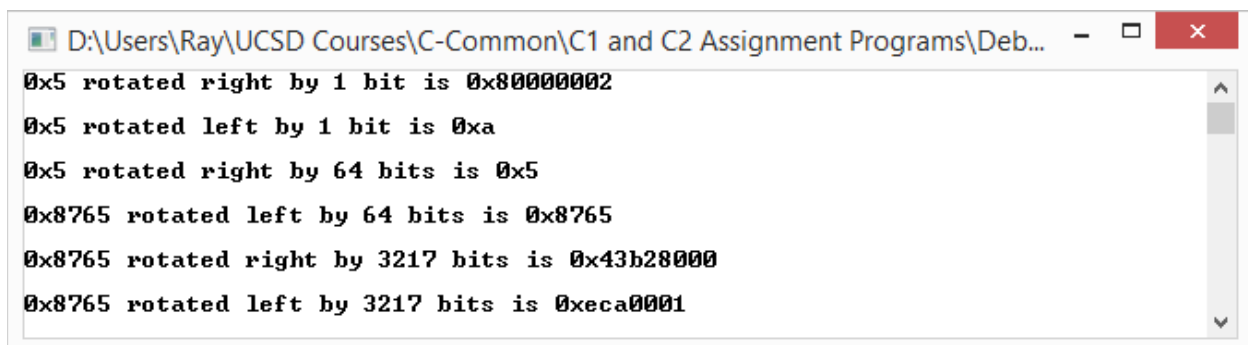
----- EXERCISE CONTINUES ON NEXT PAGE -----
```

```

1
2 ***** FILE C2A2E2_Rotate.cpp *****
3 //
4 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
5 //
6 // This file contains function Rotate, which returns its first parameter
7 // rotated by the number of bits specified by its second parameter.
8 //
9
10 int CountIntBitsF();
11
12 //
13 // Return the value resulting from rotating the pattern in <object> by the
14 // number of bit positions specified by <count>. If <count> is positive
15 // rotation will be to the right; if <count> is negative rotation will be
16 // to the left. The Rotate function exploits the fact that a left rotation
17 // by <count> is equivalent to a right rotation by the total bit-width
18 // of the object minus <count>. Note that the result of shifting an object
19 // by a negative amount or by an amount greater than the number of bits
20 // in the object is undefined. The modulus operator is used to prevent
21 // the second case from happening.
22 //
23 unsigned Rotate(unsigned object, int count)
24 {
25     int bits = CountIntBitsF();
26
27     // Get (abs(count) % bits) in case (count >= bits) in object.
28     if (count < 0)
29         count = bits - (-count % bits);
30     else
31         count %= bits;
32
33     return((object >> count) | (object << (bits - count)));
34 }

```

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



```

D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...
0x5 rotated right by 1 bit is 0x80000002
0x5 rotated left by 1 bit is 0xa
0x5 rotated right by 64 bits is 0x5
0x8765 rotated left by 64 bits is 0x8765
0x8765 rotated right by 3217 bits is 0x43b28000
0x8765 rotated left by 3217 bits is 0xeca0001

```

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

Memory Addresses		Stack Values	Description	
Relative	Absolute			
BP+??	??	??	??	startup Stack Frame
BP+??	FA9h	??	??	
BP+Ah	FA6h	??	Return Object (int)	main Stack Frame
BP+5h	FA1h	??	Function Return Address	
BP	F9Ch	??	Previous Frame Address	
BP-4h	F98h	??	val	
BP+Ah	F94h	??	Return Object (long)	Ready Stack Frame
BP+5h	F8Fh	AB4h	Function Return Address	
BP	F8Ah	F9Ch	Previous Frame Address	
BP-4h	F86h	??	res	
BP+12h	F82h	??	Return Object (long)	gcd Stack Frame 1
BP+Eh	F7Eh	96	y	
BP+Ah	F7Ah	128	x	
BP+5h	F75h	108h	Function Return Address	
BP	F70h	F8Ah	Previous Frame Address	
BP+12h	F6Ch	??	Return Object (long)	gcd Stack Frame 2
BP+Eh	F68h	32	y	
BP+Ah	F64h	96	x	
BP+5h	F5Fh	7C0h	Function Return Address	
BP	F5Ah	F70h	Previous Frame Address	
BP+12h	F56h	??	Return Object (long)	gcd Stack Frame 3
BP+Eh	F52h	0	y	
BP+Ah	F4Eh	32	x	
BP+5h	F49h	7C0h	Function Return Address	
BP	F44h	F5Ah	Previous Frame Address	

BP

F44h

→

BP

F44h

SP

F44h

→

BP

F44h

No C2A2E3 Screen Shot – Not a program

Exercise 4 (6 points – C++ Program)

```
1 ***** FILE C2A2E4_OpenFile.cpp *****
2
3 //
4 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
5 //
6 // This file contains function OpenFile, which opens for input the file
7 // specified by its first parameter using the object specified by its
8 // second parameter.
9 //
10 //
11
12 #include <fstream>
13 #include <iostream>
14 #include <cstdlib>
15
16 //
17 // Open the file named in <fileName> using the object referenced by
18 // <inFile>. If it fails display an error message and terminate the
19 // program with an error code.
20 //
21 void OpenFile(const char *fileName, std::ifstream &inFile)
22 {
23     // Open file for read only.
24     inFile.open(fileName);
25     // If open fails print an error message and terminate with an error code.
26     if (!inFile.is_open())
27     {
28         std::cerr << "File \"" << fileName << "\" didn't open.\n";
29         std::exit(EXIT_FAILURE);
30     }
31 }
32
33 ***** FILE C2A2E4_Reverse.cpp *****
34 //
35 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
36 //
37 // This file contains functions:
38 //   IsSep: Determines if its parameter represents a separator;
39 //   Reverse: Recursively reverses and displays characters read from a file;
40 //
41
42 #include <cctype>
43 #include <fstream>
44 #include <iostream>
45
46 const int CAPITALIZATION_LEVEL = 1; // recursive level for capitalization
47
48 //
49 // Test if the value in <ch> is one of the separators required in this
50 // exercise. Return true if so and false if not.
51 //
52 inline bool IsSep(int ch)
53 {
54     //
55     // Whitespace is most appropriately checked by isspace, not by
56     // checking individual specific characters.
57     //
```

```

1  return(std::isspace(ch) || ch == '.' || ch == '?' || ch == '!' ||
2      ch == ',' || ch == ':' || ch == ';' || ch == EOF);
3  }
4
5  //
6  // As each recursive level of the function is entered one input character is
7  // read and stored in <thisChar>. This continues until a separator is
8  // encountered, which is then stored in <thisSeparator>. The function then
9  // begins returning <thisSeparator> back through all levels of recursion.
10 // After each return the character in <thisChar> is displayed and, if at
11 // the level specified by CAPITALIZATION_LEVEL, is also capitalized.
12 // The function then returns the separator to the caller.
13 //
14 int Reverse(std::ifstream &inFile, const int level)
15 {
16     int thisChar = inFile.get();           // get next character...
17     if (IsSep(thisChar))                  // ...if character is separator
18         return thisChar;                  // ...then return it
19
20     int thisSeparator = Reverse(inFile, level + 1); // get next character
21     // Print character, capitalizing if at level CAPITALIZATION_LEVEL.
22     if (level == CAPITALIZATION_LEVEL)
23         std::cout.put((char)std::toupper(thisChar));
24     else
25         std::cout.put((char)thisChar);
26
27     return thisSeparator;                  // return separator
28 }

```

C2A2E4 Screen Shot

The code is a recursive function to reverse a string and capitalize it at a specific level. The code is written in a style where comments are on the same line as the code, and some lines are commented out with /* and */. The code is as follows:

```

txeT eliF elpmxE: hcaE droceR sI enO eniL fO ibrA,T,yraR htgneL sdne< htiW >'n ^
\':
tahW! rehtonA sselesU, diputS, dnA yrassecennU margorP?
seY; tahW esle?: yrT tupnI R.E.D:I;R?E!n\o/i*t=C. */./ * /.!?,;:/*+=
*/ gnisU C /*
enifed# ON_CER 9 */ rebmuN fO droceR oT daeR /*
rahC HGUONE_GIB[pmET?]; */ yarrA roF loH;gniD ynA droceR /*
tnI I;
roF i< = 0; I < ON_CER; >i++ */ roF tsriF dedeennU sdroceR /*
fI csf<?pf<fnA, >"c*%ln^[*%" == >FOE < */ daeR dnA worhT yawA /*
detcepxenU"<stupF!detcE "n\FOE, >rredTS; */ erehT sI oN ceR.drO ON_C
ER /*
>ERULIAF_TIXE<tixE; */ rorrE tixE /*
>
fI gf<:pmet<stE, >pmet<foeziS, >pF == >LLUN < */ daeR droceR ON_CER /*
pxenU"<stupF!detcE "n\FOE, >rredTS; */ erehT sI oN ceR.drO ON_CER /*
>ERULIAF_TIXE<tixE; */ rorrE tixE /*
>
*/ tA sihT tnioP (?) droceR # ON_CER sI nI pmeT. /*

```