```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E1_CountBitsM.h
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E1_CountIntBitsF.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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)
{
   //
   // 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.
   //
   int count = 0;
   for (unsigned pattern = 1u; pattern; pattern <<= 1)
      ++count;
   return count;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E2_CountIntBitsF.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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()
{
   //
   // 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.
   //
   int count = 0;
   for (unsigned pattern = 1u; pattern; pattern <<= 1)
      ++count;
   return count;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E2_Rotate.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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 whereas if negative it 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 width of the object minus <count>.
// Since the result of shifting an object by a negative amount or by an
// amount greater than or equal to the number of bits in the object is
// undefined, the modulus operator is used to prevent that from happening.
//
unsigned Rotate(unsigned object, int count)
{
   int bits = CountIntBitsF();

   // FYI
   // Although the single line of code
   //    count = (count % bits + bits) % bits;
   // could be used to determine the adjusted value of <count> instead of the
   // next 4-lines of if/else code, it is less efficient because a modulo
   // operation will be performed twice rather than just once.
   //
   if (count < 0)
      count = bits - (-count % bits);
   else
      count %= bits;

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

Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming II
Section 888888, Ray Mitchell
July 7, 2020
C2A2E3_StackFrames.pdf
Recursive Stack Frames

1. The calling convention is: "C"
2. The first call to the gcd function is: gcd(480, 360)
3. An 'h' suffix indicates a hexadecimal value.
4. ?? indicates the value is unavailable or not assigned until the function returns.
5. The byte size is for your information and is not required in your figure.

```
Rel      Abs   Stack         Description           Byte
Adr      Adr   Value                               Size
------------------------ startup -------------------------
BP+7h    B8Fh  1005h   Function Return Address   (7 bytes)
BP       B88h     0h   Previous Frame Address    (7 bytes)
------------------------- main --------------------------
BP+Eh    B83h    ??    Return Object             (5 bytes)
BP+7h    B7Ch    ??    Function Return Address   (7 bytes)
BP       B75h   B88h   Previous Frame Address    (7 bytes)
BP-5h    B70h    ??    result                    (5 bytes)
------------------------- ready -------------------------
BP+Eh    B68h    ??    Return Object             (8 bytes)
BP+7h    B61h   62Dh   Function Return Address   (7 bytes)
BP       B5Ah   B75h   Previous Frame Address    (7 bytes)
BP-4h    B56h    ??    temp                      (4 bytes)
------------------------- gcd 1 -------------------------
BP+1Ah   B51h    ??    Return Object             (5 bytes)
BP+16h   B4Dh   360    y                         (4 bytes)
BP+Eh    B45h   480    x                         (8 bytes)
BP+7h    B3Eh   3F0h   Function Return Address   (7 bytes)
BP       B37h   B5Ah   Previous Frame Address    (7 bytes)
------------------------- gcd 2 -------------------------
BP+1Ah   B32h    ??    Return Object             (5 bytes)
BP+16h   B2Eh   120    y                         (4 bytes)
BP+Eh    B26h   360    x                         (8 bytes)
BP+7h    B1Fh   45Dh   Function Return Address   (7 bytes)
BP       B18h   B37h   Previous Frame Address    (7 bytes)
------------------------- gcd 3 -------------------------
BP+1Ah   B13h    ??    Return Object             (5 bytes)
BP+16h   B0Fh     0    y                         (4 bytes)
BP+Eh    B07h   120    x                         (8 bytes)
BP+7h    B00h   45Dh   Function Return Address   (7 bytes)
BP & SP  AF9h   B18h   Previous Frame Address    (7 bytes)
----------------------------------------------------------
```

```cpp
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E4_OpenFile.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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);
   }
}
```

```cpp
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A2E4_Reverse.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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 CAP_LEVEL_A = 1; // recursive level for capitalization
const int CAP_LEVEL_B = 3; // 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 a capitalization level.
   if (level == CAP_LEVEL_A || level == CAP_LEVEL_B)
      thisChar = std::toupper(thisChar);
   std::cout.put((char)thisChar);

   return thisSeparator;                            // return separator
}
```