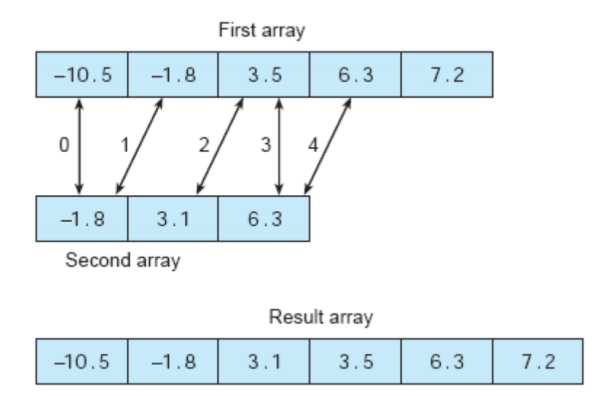
1. **(Reversing the order of an integer’s bit)** Write a program that reverses the order of the bits in an unsigned int value. The program should input the value from the user and call function reverseBits to print the bits in reverse order. Print the value in bits both before and after the bits are reversed to confirm that the bits are reversed properly.
2. **(Packing Characters into an Integer)** The left-shift operator can be used to pack two character values into an unsigned integer variable. Write a program that inputs two characters from the keyboard and passes them to function packCharacters. To pack two characters into an unsigned integer variable, assign the first character to the unsigned variable, shift the unsigned variable left by 8 bit positions and combine the unsigned variable with the second character using the bitwise inclusive OR operator. The program should output the characters in their bit format before and after they are packed into the unsigned integer to prove that the characters are in fact packed correctly in the unsigned variable
3. Write a function that will merge the contents of two sorted (ascending order) arrays of type double values, storing the result in an array output parameter (still in ascending order). The function should not assume that both its input parameter arrays are the same length but can assume that one array does not contain two copies of the same value. The result array should also contain no duplicate values.

Assume the max length of the first and second array is 20. Hint: When one of the input arrays has been exhausted, do not forget to copy the remaining data in the other array into the result array. Test your function with cases in which (1) the first array is exhausted first, (2) the second array is exhausted first, and (3) the two arrays are exhausted at the same time (i.e., they end with the same value). Remember that the arrays input to this function must already be sorted.



Test your function by the following cases:

1. 1st array: -12 -7.65 -2.4 0 1.18 7.92 12.888 24.23

2nd array: -13.5 -1.423 0.023 1.0112 2.5

1. 1st array: -256.44 -101.393 -33.2356 -1.123 6.8

2nd array: -4.543 -1.123 64.8 140.231 236.22 484.93 5893.231 32233.2

1. 1st array: -23.2 -14.8 34.69 39.88 57.45

2nd array: -16.83 -6.34 8.893 15.223 60.54

1. A barcode scanner for Universal Product Codes (UPCs) verifies the 12 digit code scanned by comparing the code’s last digit (called a check digit) to its own computation of the check digit from the first 11 digits as follows:
2. Calculate the sum of the digits in the odd-numbered positions (the first, third,…, eleventh digits) and multiply this sum by 3.
3. Calculate the sum of the digits in the even-numbered positions ( the second, fourth,…, tenth digits) and add this to the previous result.
4. If the last digit of the result from step ii is 0, then 0 is the check digit. Otherwise, subtract the last digit from 10 to calculate the check digit.
5. If the check digit matches the final digit of the 12 digit UPC, the UPC is assumed correct.

Write a program that prompts the user to enter the 12 digits of a barcode separated by spaces. The program should store the digits in an integer array, calculate the check digit, and compare it to the final barcode digit. If the digits match, output the barcode with the message “validated.” If not, output the barcode with the message “error in barcode.”