ERROR DETECTION: PARITY BITS AND CHECK DIGITS

Robert P. Webber, Longwood University

When data is entered into a computer, or when data is sent over transmission lines or cables within a computer, inadvertent errors can occur. Minute particles of dirt or grease can corrupt data on a disk, for example. Static on a telephone line can introduced errors into transmitted data.

It is very important to be sure that data has not been corrupted. **Error detection** checks for errors that occur in the transmission or storage of data. **Error correction** determines that an error has occurred and tries to fix the mistake.

A simple error detection method is based on the principle that if each bit pattern being manipulated as an odd numbers of 1s, and a pattern is detected that has an even number of 1s, then an error must have occurred. A **parity bit** is an extra bit that is associated with a word of storage. The value of 1 or 0 is assigned to the parity bit to make the total number of 1s in the word odd if **odd parity** is used, and even if **even parity** is used.

For example, the ASCII code for 'A' is 0100 0001. Using odd parity, it is 1 0100 0001. The extra bit is the parity bit, and it is set to 1 because 0100 0001 has an even number of 1s. On the other hand, the ASCII code for 'C' is 0100 0011. This code already has an odd number of 1s, so the representation using odd parity would be 0 0100 0011.

With odd parity, an error condition is indicated by any nine bit pattern with an even number of 1s.

Modern computer memories use built-in parity bits. We think of the basic unit of memory as the byte, consisting of eight bits. In reality, it is a byte plus a parity bit, or nine bits in all. Either odd parity or even parity can be used.

When something is stored in memory, the operating system sets the parity bit. Suppose it uses odd parity. When a value is retrieved from memory, the system checks its parity. If the parity is still odd, the system returns the value. If it is even, the system may return the value, but it will be accompanied by a warning that the value may have been corrupted.

The primary advantages of parity are its simplicity and ease of use. Its primary disadvantage is that it may fail to catch errors. If two data bits are corrupted, for instance, parity will not detect the error.

Here's another way to look at parity bits. Recall the *xor* operation:

x	*y*	*xor*
0	0	0
0	1	1
1	0	1
1	1	0

Notice that x *xor* y is 1 when exactly one of x, y is 1 and 0 otherwise. By extension, the *xor* of a bunch of bits of 1 precisely when there are an odd number of 1s and 0 otherwise. This means that *xor* can be use to set parity. Just *xor* the bits in sequence, using the result from one *xor* as one of the operands to the next *xor*.

For odd parity, if the result of the final *xor* is 1, set the parity bit to 0 (since there are already an odd number of 1s). If the result is 0, set the parity bit to 1. Reverse the assignment for even parity.

The byte plus the parity bit is transmitted. The receiver puts the entire transmission (byte plus the parity bit) through *xor*. If we are using odd parity, and the resulting *xor* is 0, we know an error occurred, since 0 indicates an even number of 1s.

A **check digit** is a variation on the parity bit scheme. In its simplest form, it stores a number, and then stores the units digit of the sum of the digits of the original number. For example, suppose the number is 121208. The sum of the digits is 14, so we would store 4 in addition to the number.

This method has the same advantages and disadvantages as parity bits. A single error probably will be detected, but several errors may not be.

A variation of this technique was used in the early days of personal computers. No Internet was available to share programs in those days, and enthusiasts manually entered machine language numerical codes for programs. These were rows and rows of numbers, and it was easy to make a typing mistake. The source code would have a check digit at the end of each row, and the computer checked the sum of the data entered against the check digit to detect an error.

Many credit card companies use a check digit in their account numbers. If you have a card with a 16 digit number, for example, the last digit is probably a check digit calculated according to an algorithm called the **Luhn formula**, named for IBM scientist Hans Peter Luhn, who invented it in 1954. Here is how it works.

The check digit is the last digit on the right. Start with this digit.

1. Counting from the check digit and moving to the left, double the value of every second digit.
2. Sum the digits of the products together with the undoubled digits from the original number.

3. Set the check digit so the resulting sum ends in 0 (that is, so the resulting sum is a multiple of 10).

For example, suppose the account number is 2413 0586 2276 081*x* , where *x* is the check digit. (I've grouped the digits by fours for ease of reading.) Calculate the check digit by following the steps of the Luhn algorithm.

Original number, with every second digit highlighted, starting on the right:

2 4 ***1*** 3 ***0*** 5 ***8*** 6 ***2*** 2 ***7*** 6 ***0*** 8 ***1*** *x*

Double the highlighted digits:

4 ***2*** ***0*** ***16*** ***4*** ***14*** ***0*** ***2***

Add the unhighlighted digits of the original number and the digits of the doubled highlighted numbers:

4 + 4 +***2*** + 3 + ***0*** + 5 + ***1 + 6*** + 6 + ***4*** + 2 + ***1 + 4*** + 6 + ***0*** + 8 + ***2*** + *x* = 58 + *x*

To make the sum a multiple of 10, set $x = 2$. The full account number is 2413 0586 2276 0812 .

To check whether a given account number is invalid, do the Luhn calculations. If the resulting sum does not end in 0, the number is not valid. For instance, check the account number 9413 0025 1616 2853 .

Original number, with every second digit highlighted, starting on the right:

9 4 ***1*** 3 ***0*** 0 ***2*** 5 ***1*** 6 ***1*** 6 ***2*** 8 ***5*** 3

Double the highlighted digits:

18 ***2*** ***0*** ***4*** ***2*** ***2*** ***4*** ***10***

Add the unhighlighted digits of the original number and the digits of the doubled highlighted numbers:

1 + 8 + 4 +***2*** + 3 + ***0*** + 0 + ***4*** + 5 + ***2*** + 6 + ***2*** + 6 + ***4*** + 8 + ***1 + 0*** + 3 = 59

Since 59 does not end in 0, the number is invalid.

The Luhn algorithm does not catch all possible errors. It was intended to detect obvious mistakes, such as reversing two digits when entering a number, or entering a digit incorrectly. A clever crook could easily devise a number that would pass the Luhn test! Nevertheless, most credit card companies use it when assigning account numbers, probably because computer programs can check it quickly. It provides a valuable first line of defense against invalid account numbers.

EXERCISES

1. Fill in the blank with the value of the parity bit using odd parity.

   a. __ 01000110

   b. __ 00100000

   c. __00100111

   d. __10000111

2. Repeat exercise 1, but use even parity.

3. The following bytes were originally written using odd parity. In which can you be sure that an error has occurred?

	Parity bit	Byte
a.	1	10010001
b.	1	01101100
c.	0	00111010
d.	0	01101011

4. Could an error have occurred in bytes other than the ones you detected in exercise 3? Explain.

5. The following bytes were originally written using even parity. In which can you be sure that an error has occurred?

	Parity bit	Byte
a.	0	01101000
b.	1	00101100
c.	0	10101100
d.	1	01111110

6. Could there be errors in bytes other than the ones you detected in exercise 5? Explain.

7. Write the check digit for each number in the blank.

   a. 32767__
   b. 296__

8. Write the check digit for each number in the blank.

a. 2824__

b. 3210_

9. The following numbers were originally written using a check digit. In which can you be sure an error has occurred?

	Number	Check digit
a.	5432	4
b.	10687	1

10. Could there be errors in the numbers in problem 9 that you did not detect using the check digit? Explain.

11. The following numbers were originally written using a check digit. In which can you be sure an error has occurred?

	Number	Check digit
a.	314159	0
b.	217282	2

12. Could there be errors in the numbers in problem 11 that you did not detect using the check digit? Explain.

MasterCard uses 16 digit account numbers, with the 16^{th} digit being a Luhn's formula check digit. In exercises 13 and 14, determine whether the given MasterCard account number is invalid.

13. 2418 0996 3416 5066

14. 6205 1883 5462 3917

15. Choose the final digit x so that 5400 1286 9063 254x is a valid Mastercard account number.

VISA also uses 16 digit account numbers, with the 16^{th} digit being a Luhn's formula check digit. In exercises 16 and 17, determine whether the given VISA account number is invalid.

16. 3887 0290 1148 6720

17. 5019 4326 7675 2895

18. Choose the final digit x so that 1062 2833 4165 821x is a valid Visa account number.

American Express uses 15 digit account numbers, with the right-most digit being a Luhn's formula check digit. In exercises 19 and 20, determine whether the given American Express account number is invalid.

19. 234 1987 7293 4087

20. 625 6875 4980 7930

21. Choose the final digit x so that 213 0696 4405 881x is a valid American Express account number.

Diner's Club/Carte Blanche uses 14 digit account numbers, with the last digit on the right being a Luhn's formula check digit. In exercises 22 and 23, determine whether the given Diner's Club account number is invalid.

22. 62 1394 5660 8299

23. 25 5607 4396 8842

24. Choose the final digit x so that 20 1932 8745 635x is a valid Diner's Club account number.