Error Detection with CRC-16

Information Theory Lab 11

Objective

Understand the CRC (Cyclic encoding) algorithm and its usage for detection of errors, by implementing a C application to do CRC-16 computation and checking.

Theoretical notions

Basic CRC-16 algorithm

Consider a 16-bit generator polynomial (CRC-16) in binary format

$$\mathbf{g} = 10100000000000011$$

Given a block of binary data, the algorithm for CRC computation is as follows:

1. Locate the first 1 in the sequence
2. Starting from this location, perform XOR of the next data bits with **g**.
3. Repeat until all original data has been zeroed out.

The **CRC value** is the 16-bit value remaining at the end of the data.

CRC checking

To check received data for errors, perform CRC-16 again and check if the CRC value is the same.

Alternatively, if the CRC was value is appended at the end of the original file, one can perform CRC on the whole data (data with the CRC appended).

- If the resulting CRC value is 0, everything OK
- If the resulting CRC value is not 0, the data has been corrupted

Exercises

1. Write a C program that performs CRC-16 computation and checking.

   The program shall be called in two possible ways:

   a. with two arguments. In this case the program takes the first as input and produces the encoded file as output.

   `CRC16.exe original.dat encoded.dat`

   b. with one argument. In this case the program takes the encoded file as input and checks if the CRC is OK or not.

   `CRC16.exe encoded.dat`

   - The arguments are:
     - `original.dat`: the input file (original / encoded)
     - `encoded.dat` [optional]: the encoded file, with the CRC value appended
   - The program should consist of the following steps:
     - define an array g with the values 10100000000000011
     - declare one large vector of `unsigned char` for input bits
     - open the input file and read everything into the input vector
     - for every bit in the input vector
       - if the bit is 1, do XOR starting from this bit with the 17 bits in g
     - there will be 16 bits remaining at the end of the original input vector (the CRC-16 value)
     - then:
       a. If the program is called with two arguments:
          - write the vector to the output data file, including the CRC-16 value at the end
       b. If the program is called with one argument:
          - if the CRC-16 value is 0, display "File OK\n", otherwise display "Data corrupted\n"

2. Run the program (with two arguments) on a sample data file. Take the output encoded file run the program again on it (with a single argument). Observe the result.

3. Use the website `hexed.it` to introduce a few errors anywhere in the encoded file. Save the modified file and run the program on it (with a single argument). Observe the result.

Program design

1. The bit operation macro definitions are available in `bitmacros.h`.

For quick access to a bit number i in a large array, use the following macros:

```
#define VECREAD_BIT(v,i) (READ_BIT((v[(i)/8]),(i)%8))
#define VECWRITE_BIT(v,i,val) (WRITE_BIT((v[(i)/8]),(i)%8, val))
#define VECTOGGLE_BIT(v,i) (TOGGLE_BIT((v[(i)/8]),(i)%8))
```

Final questions

1. What is the size of the output file, compared to the input file?
2. What would change if we use a 32-bit generator polynomial (CRC-32)?